Convolution

FUSION

Convolutional Neural Network (CNN)

卷積神經網路

FUSION

Fully Connected Neural Network

n = 1000 x 1000 x 3 (height, weight, channels(RGB))

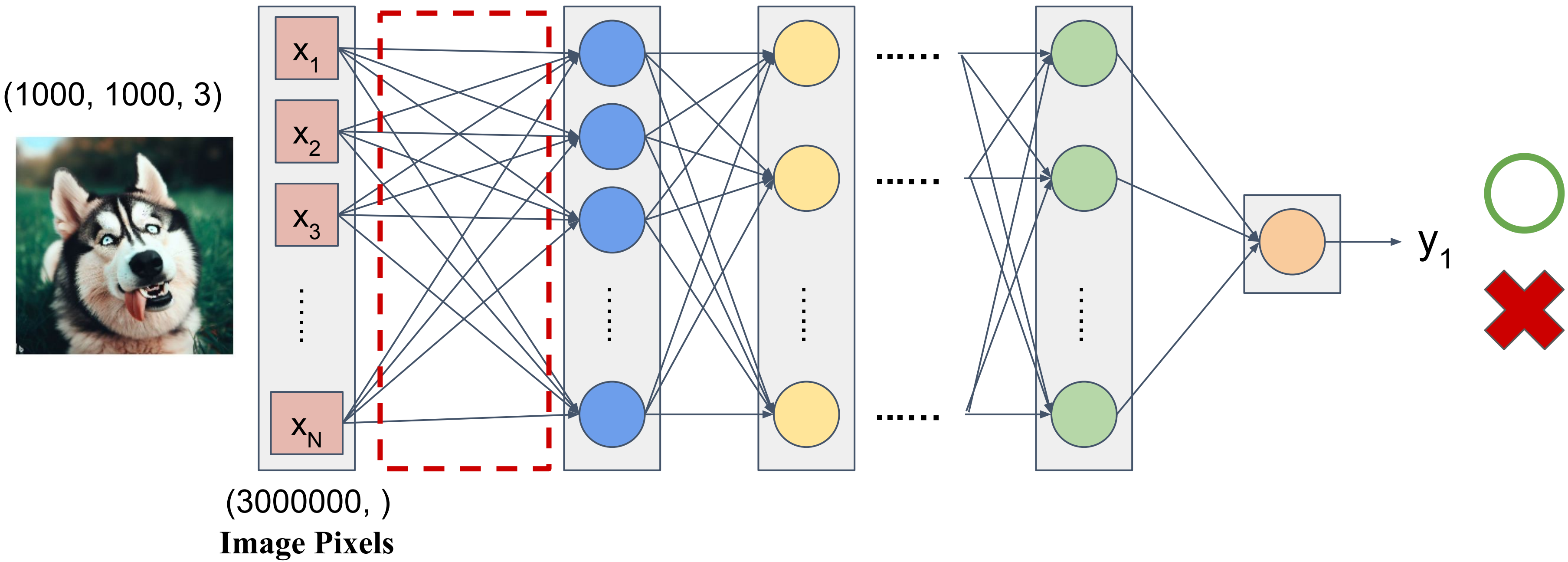


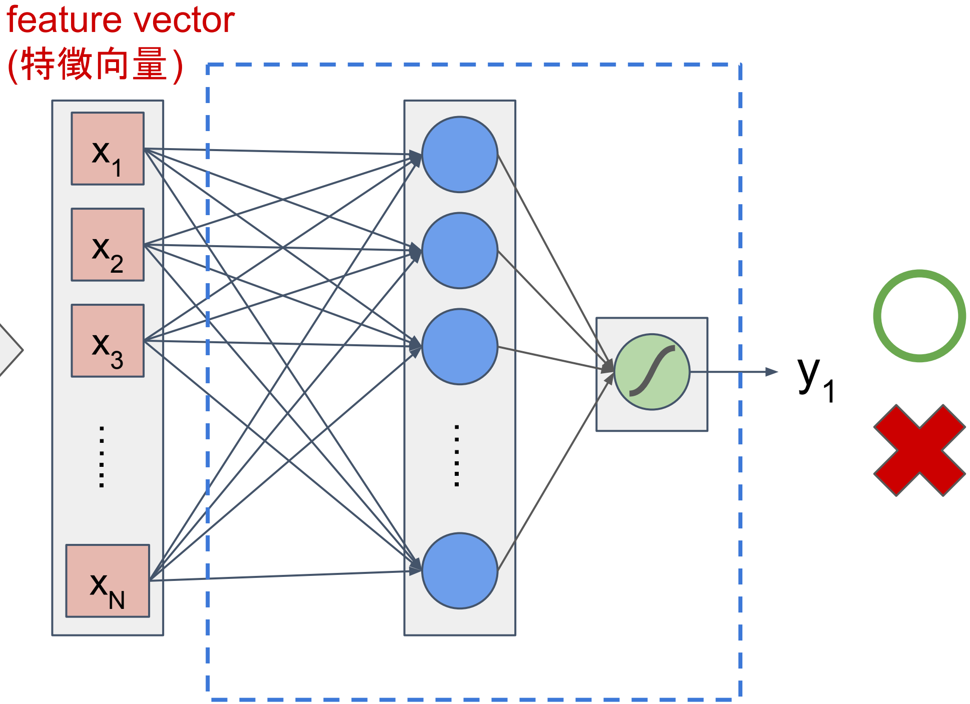
Image Recognition

Domain knowoledge

Feature Extractor

e.g.

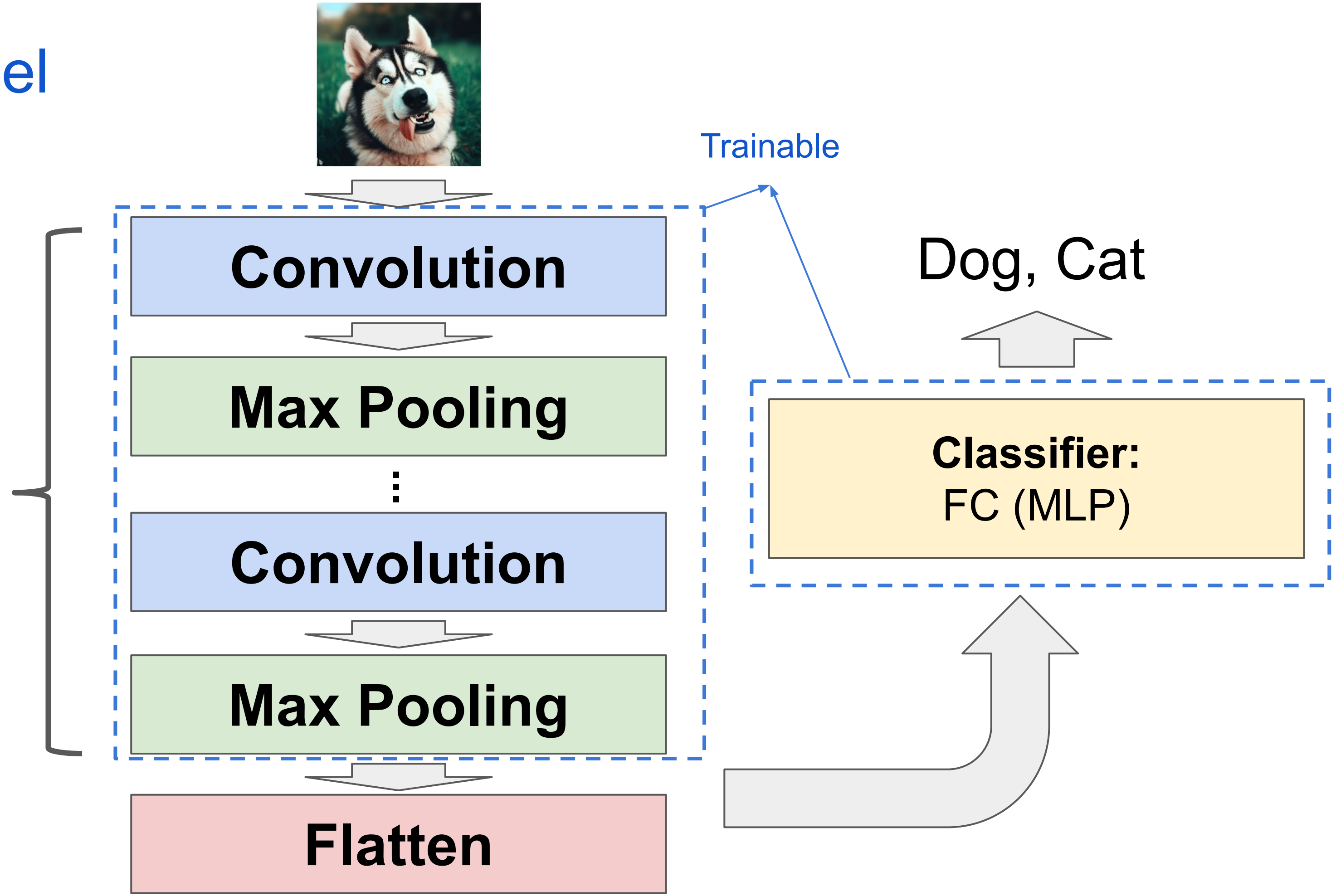
1. Color
2. Line
3. Texture
4. GLCM
5. HoG
6. Radiomics



Trainable **Classifier**

FUSION

CNN model

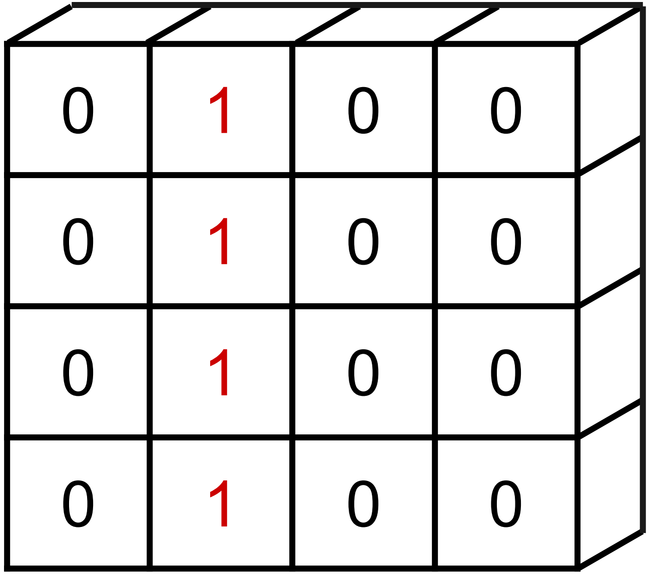


Trainable
Feature
Extractor
(Encoder)
Convolution
Max Pooling
⋮
Convolution
Max Pooling
Flatten
Dog, Cat
Classifier:
FC (MLP)

CNN - Convolution

NOTE:
- 卷積層輸入必定有通道數
- PyTorch 將通道數放在前 (channel first)

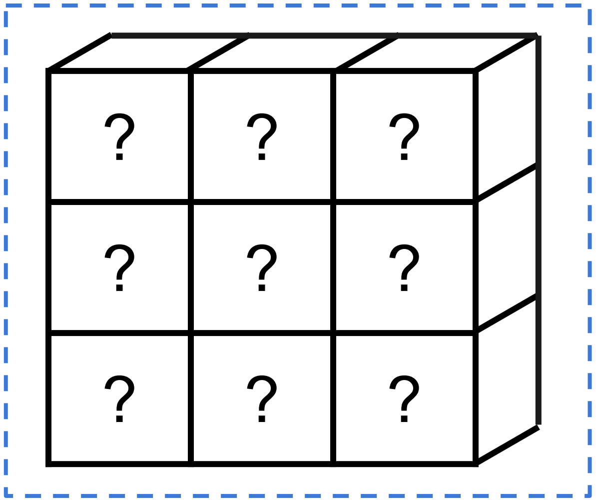
- For grayscale image (channels = 1)



4 x 4 image

(1, 4, 4) = (**C**hannels, **H**eight, **W**idth)

Trainable

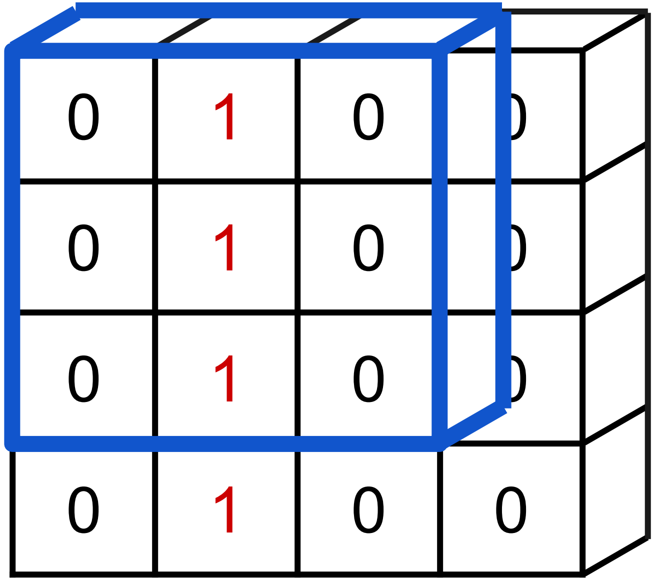


3 x 3 filter (kernel)

(1, 3, 3)

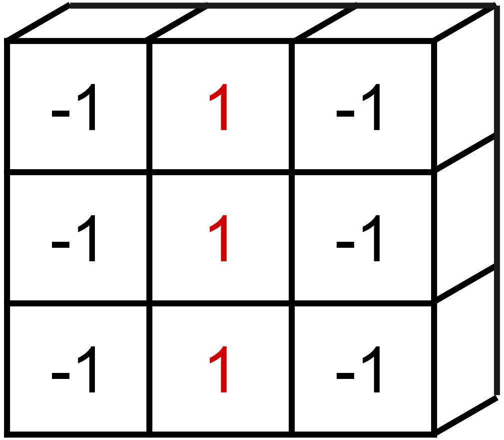
CNN - Convolution

- For grayscale image (channels = 1)



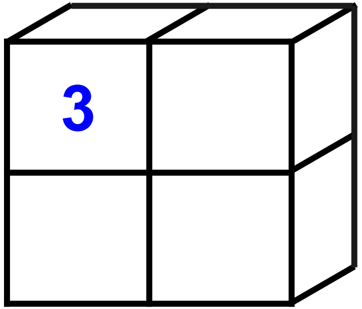
4 x 4 image

(1, 4, 4) = (**C**, **H**, **W**)



3 x 3 filter (kernel)

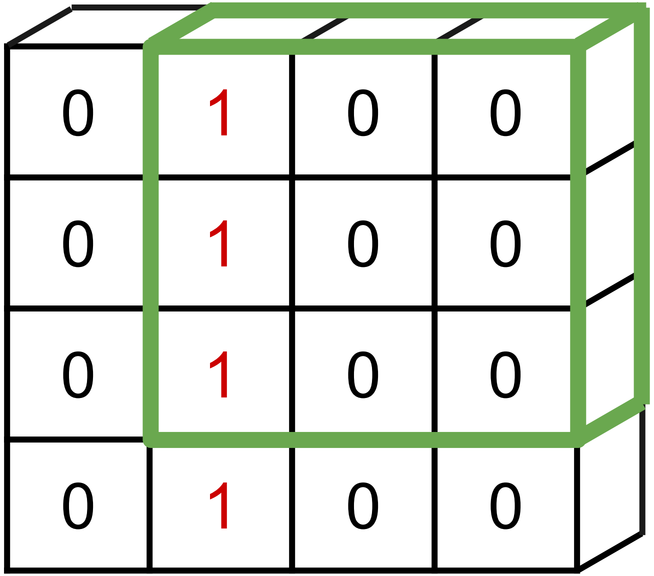
(1, 3, 3)



-1*0 + 1*1 + -1*0 +
-1*0 + 1*1 + -1*0 +
-1*0 + 1*1 + -1*0
= 3

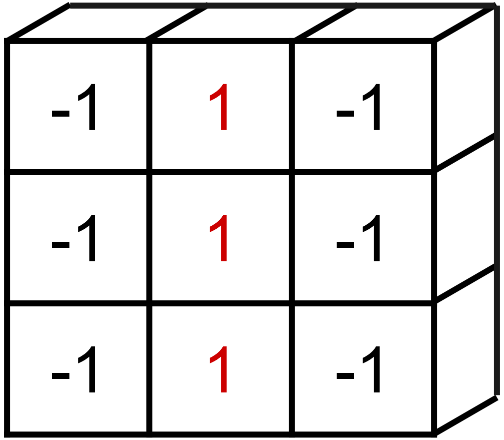
CNN - Convolution

- For grayscale image (channels = 1)



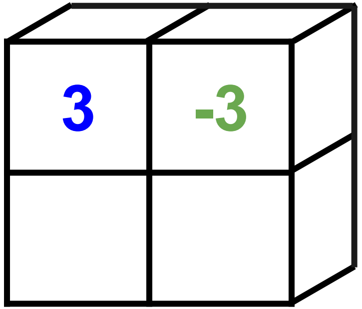
4 x 4 image

(1, 4, 4) = (**C**, **H**, **W**)



3 x 3 filter (kernel)

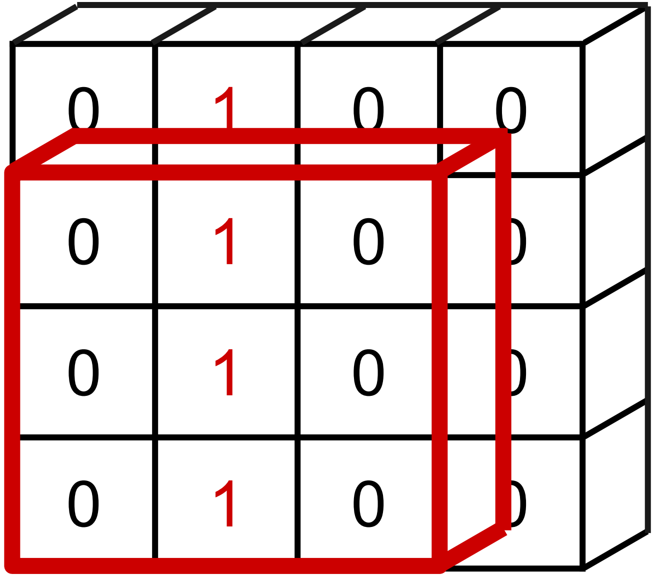
(1, 3, 3)



-1*1 + 1*0 + -1*0 +
-1*1 + 1*0 + -1*0 +
-1*1 + 1*0 + -1*0
= -3

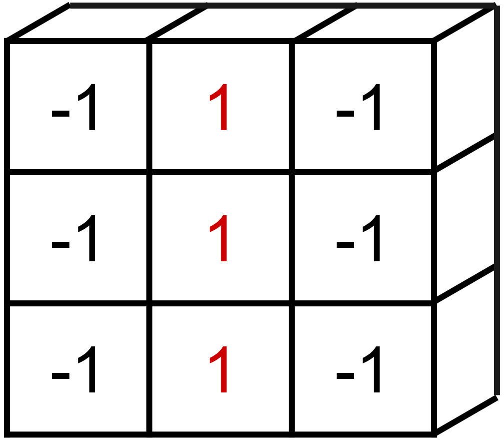
CNN - Convolution

- For grayscale image (channels = 1)



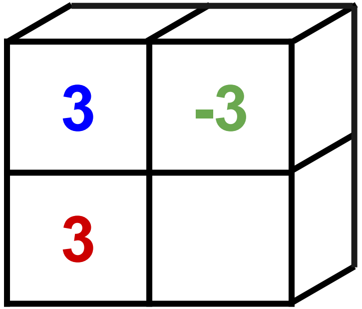
4 x 4 image

(1, 4, 4) = (**C**, **H**, **W**)



3 x 3 filter (kernel)

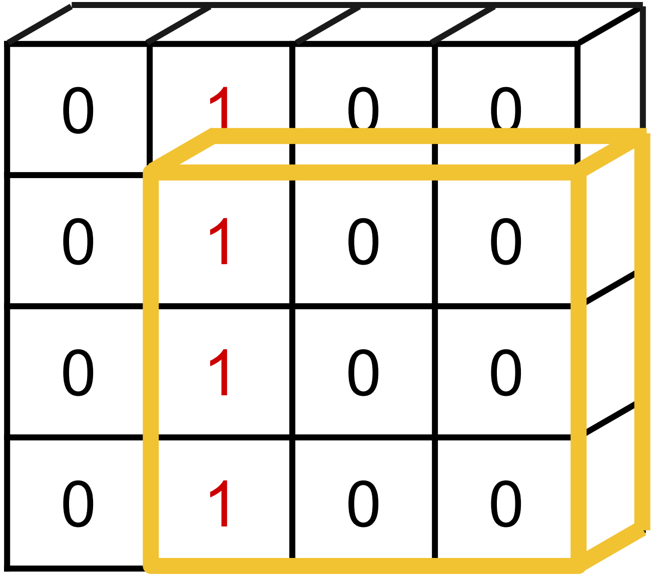
(1, 3, 3)



-1*0 + 1*1 + -1*0 +
-1*0 + 1*1 + -1*0 +
-1*0 + 1*1 + -1*0
= 3

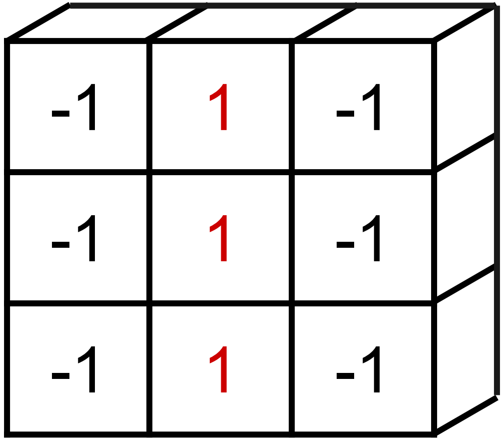
CNN - Convolution

- For grayscale image (channels = 1)



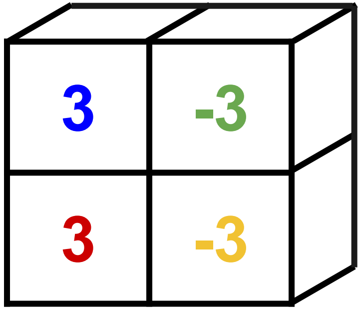
4 x 4 image

(1, 4, 4) = (**C**, **H**, **W**)



3 x 3 filter (kernel)

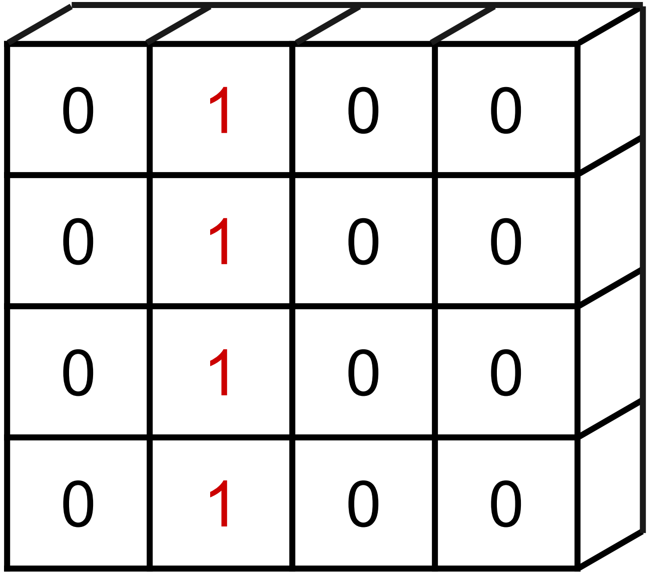
(1, 3, 3)



-1*1 + 1*0 + -1*0 +
-1*1 + 1*0 + -1*0 +
-1*1 + 1*0 + -1*0
= -3

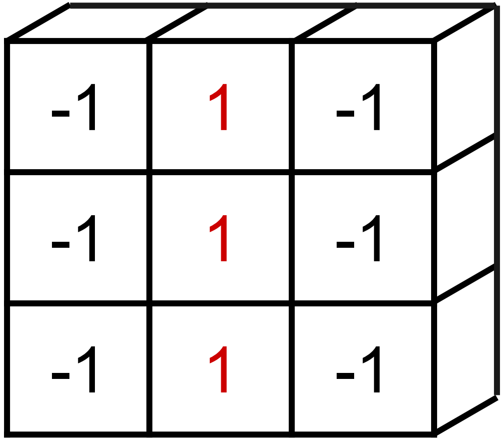
CNN - Convolution

- For grayscale image (channels = 1)



4 x 4 image

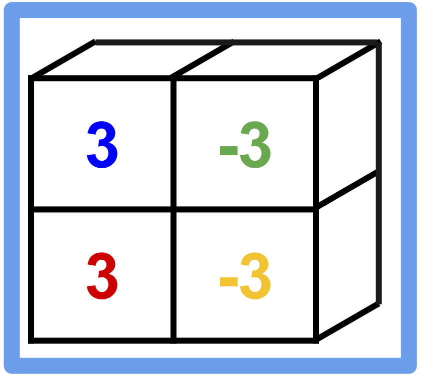
(1, 4, 4) = (**C**, **H**, **W**)



Trainable

3 x 3 filter (kernel)

(1, 3, 3)



feature map

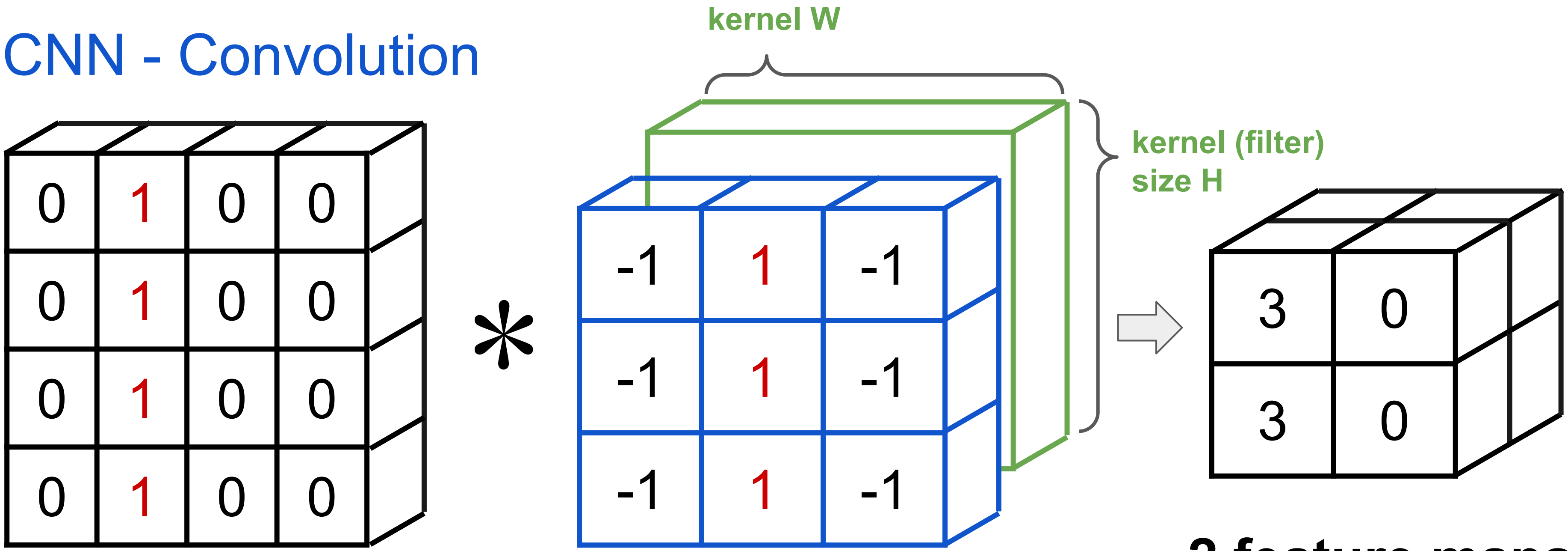
(1, 2, 2)

Information (feature)

left side with **vertical line**

FUSION

CNN - Convolution



(1, 4, 4)
(C, H, W)

2 filters

2 feature maps

(2, 2, 2,)
(C’, H’, W’)

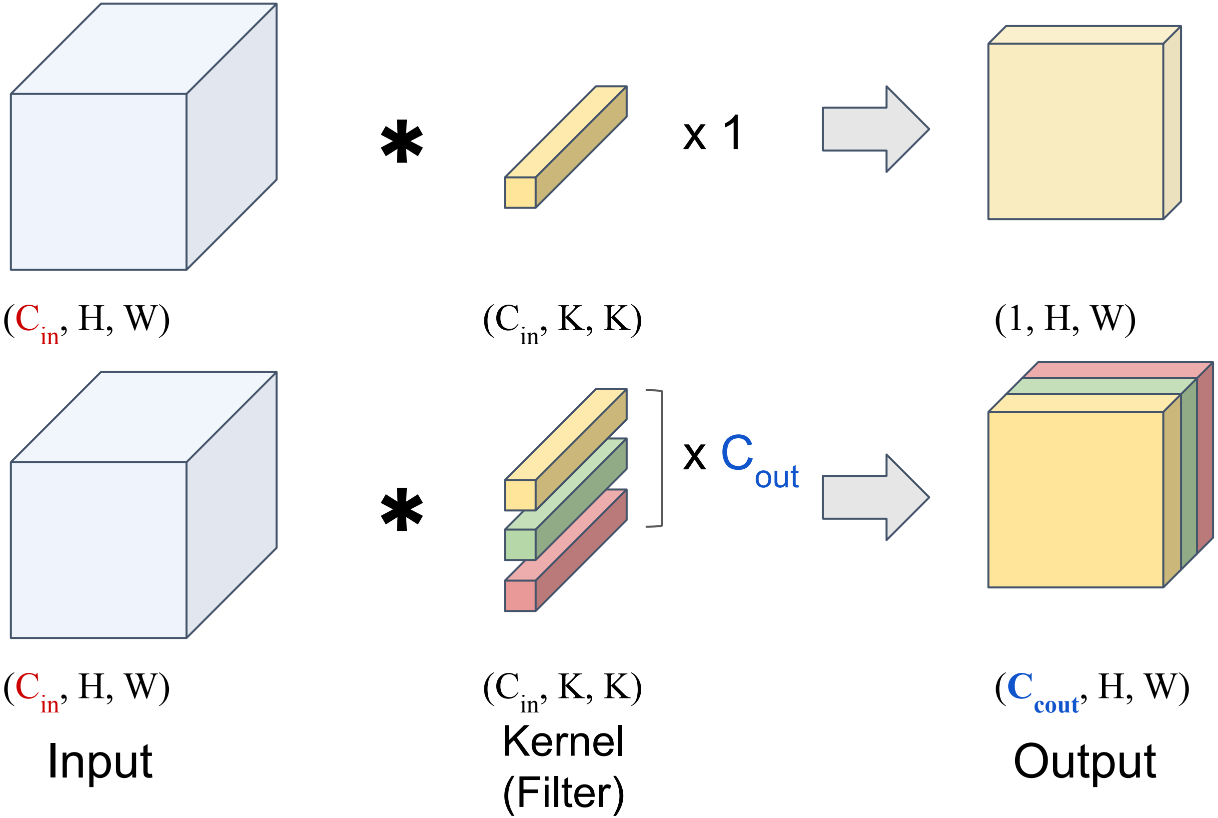
PyTorch

```
torch.nn.Conv2d(in_channels=1,
                out_channels=2,
                kernel_size=3)
```

Convolution

PyTorch

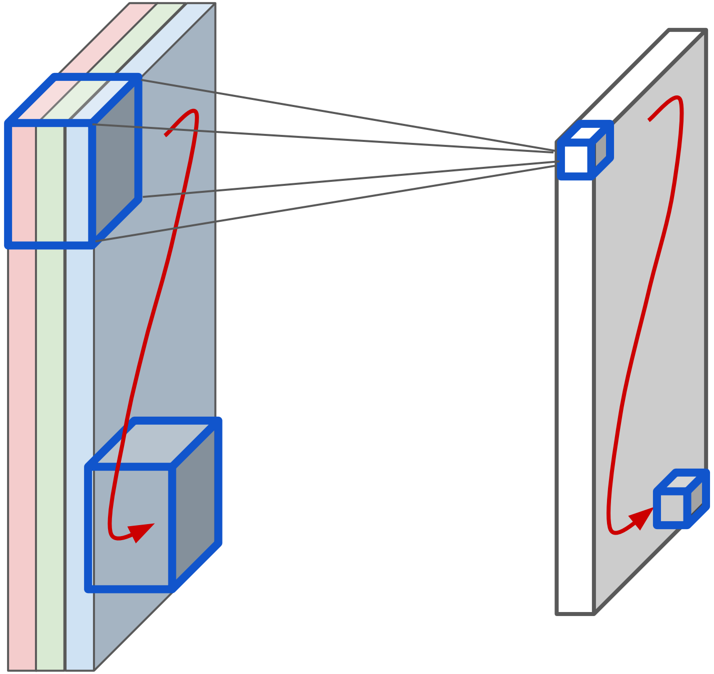
`torch.nn.Conv2d(C_in, C_out, kernel_size=K)`



FUSION

Convolution

- Filter (kernel) ***channels*** is based on input ***channels***
  - **e.g.**
    - **1 filter, kernel size=3x3**
    - **kernel shape**
      - **(Channels, kernel_H, kernel_W)**
      - **(3, 3, 3)**
- Channels
  - 1
    - Grayscale
    - CT, X-ray
    - Ultrasound
  - 3
    - Color RGB
  - N: whatever you want
    - 2: PET + CT
    - 4: RGB + Infrared
    - 4: RGB + Edge detection



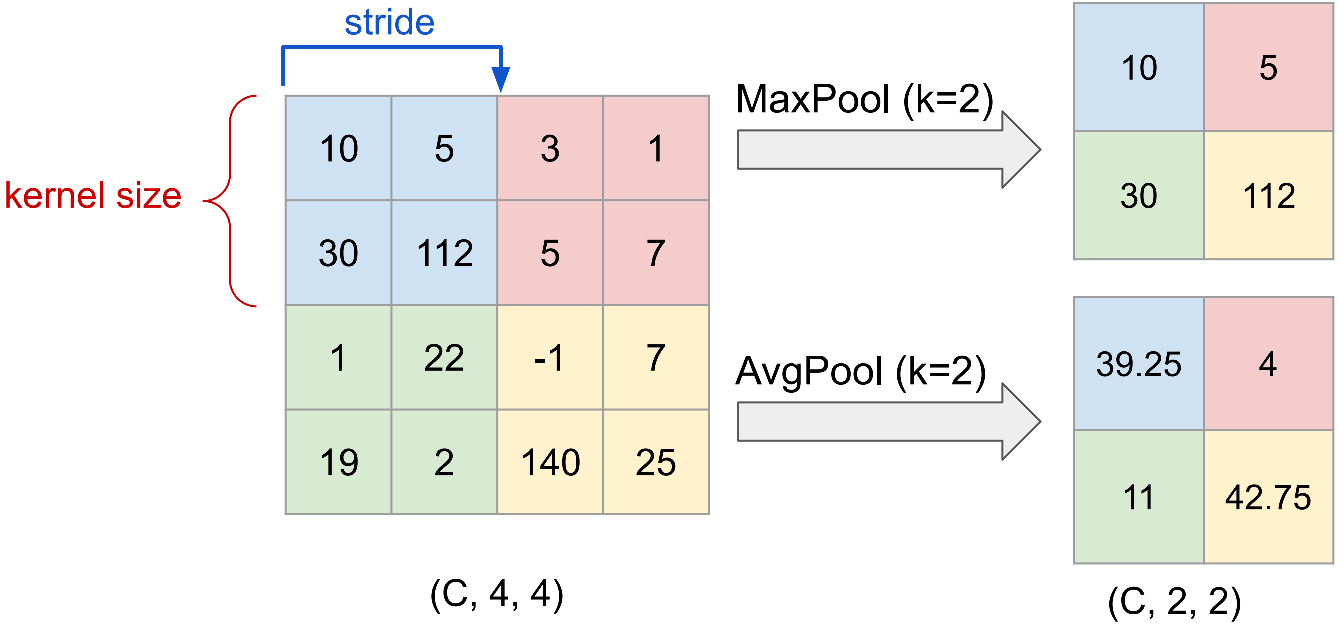
FUSION

Pooling Layer (池化層)

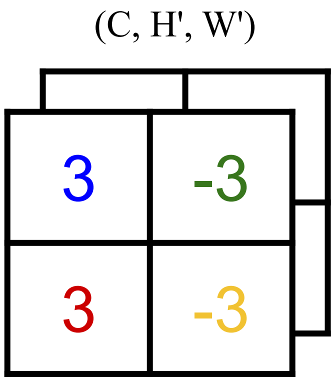
PyTorch

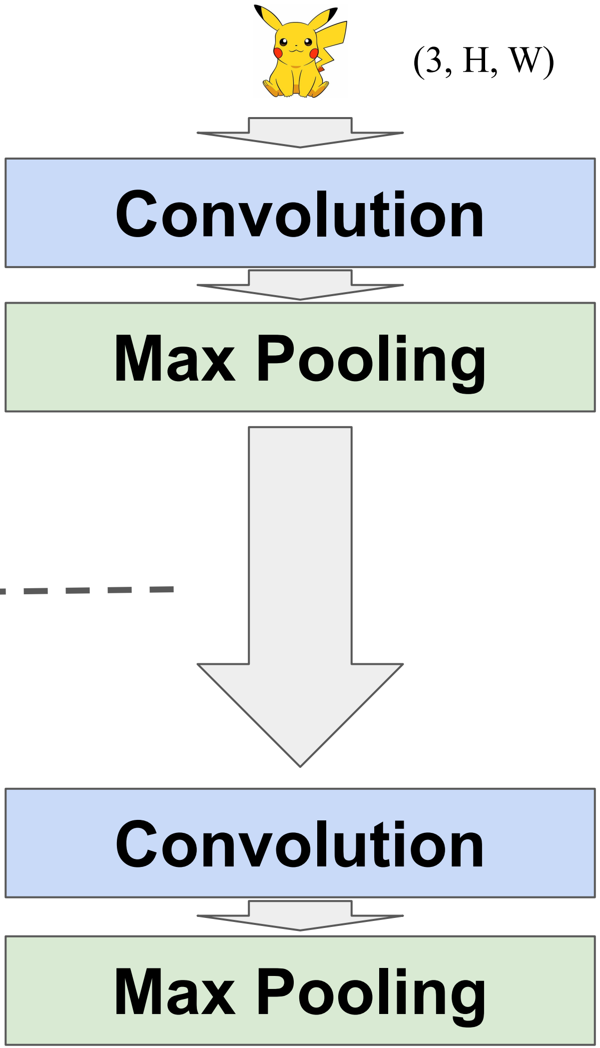
```
torch.nn.MaxPool2d(2)
torch.nn.MaxPool2d(
      kernel_size=2,
      stride=None)
torch.nn.MaxPool2d(
      kernel_size=2,
      stride=2)
torch.nn.AvgPool2d()
```

- e.g.
  - **Max** pooling
  - **Average** pooling
- Reduce size, computing complexity



After Convolution + max pooling



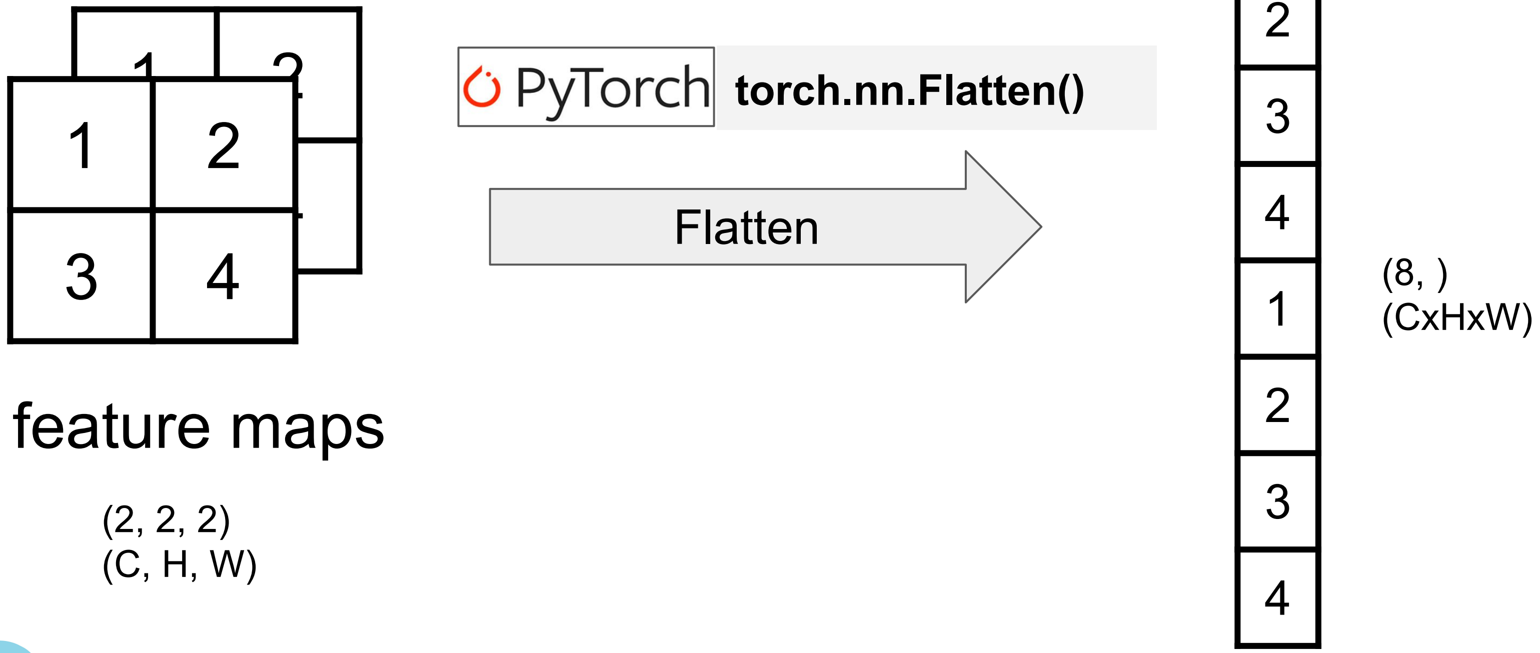


feature map 做下一層的輸入”image”

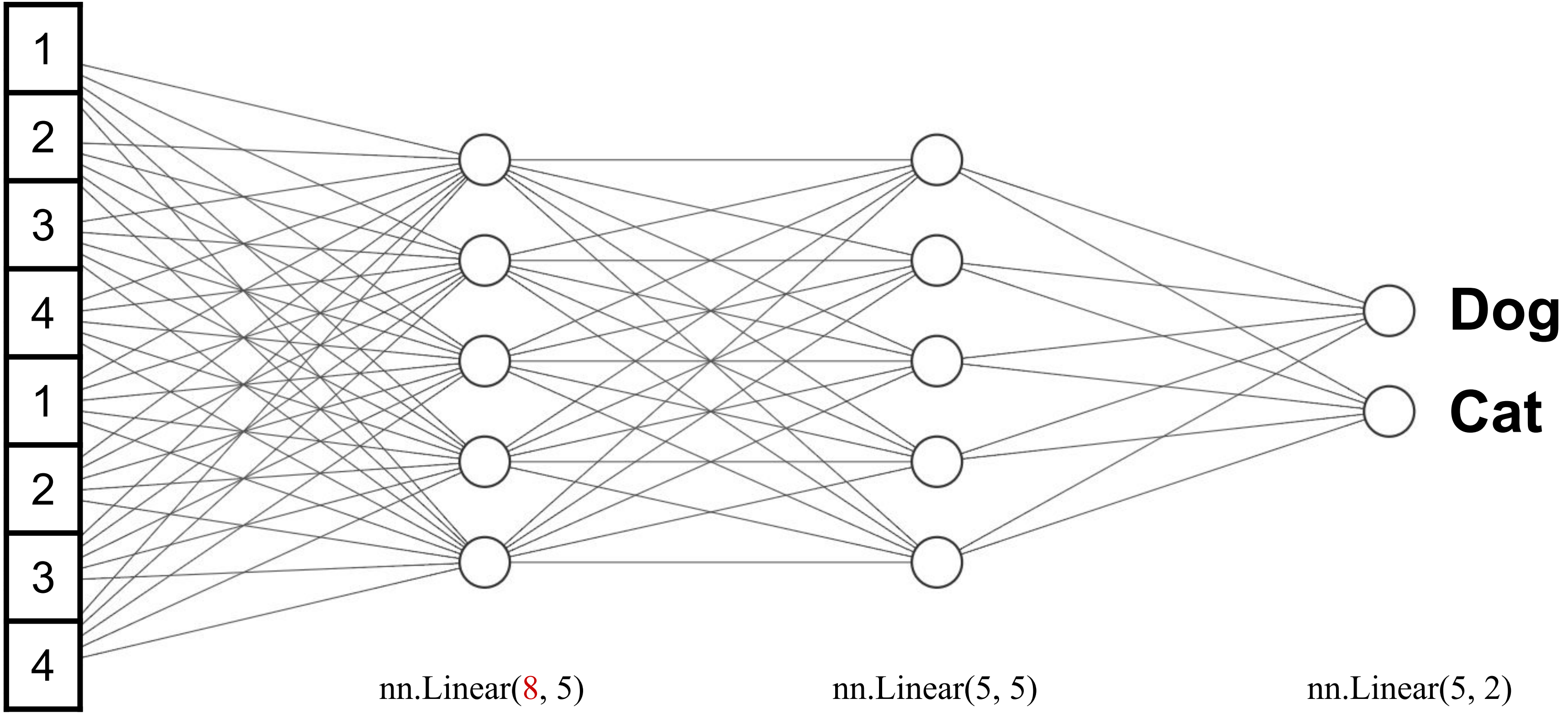
CNN - Flatten

feature vector (特徵向量)

- Flatten input tensor
- Reshape



CNN - FC



(8,)

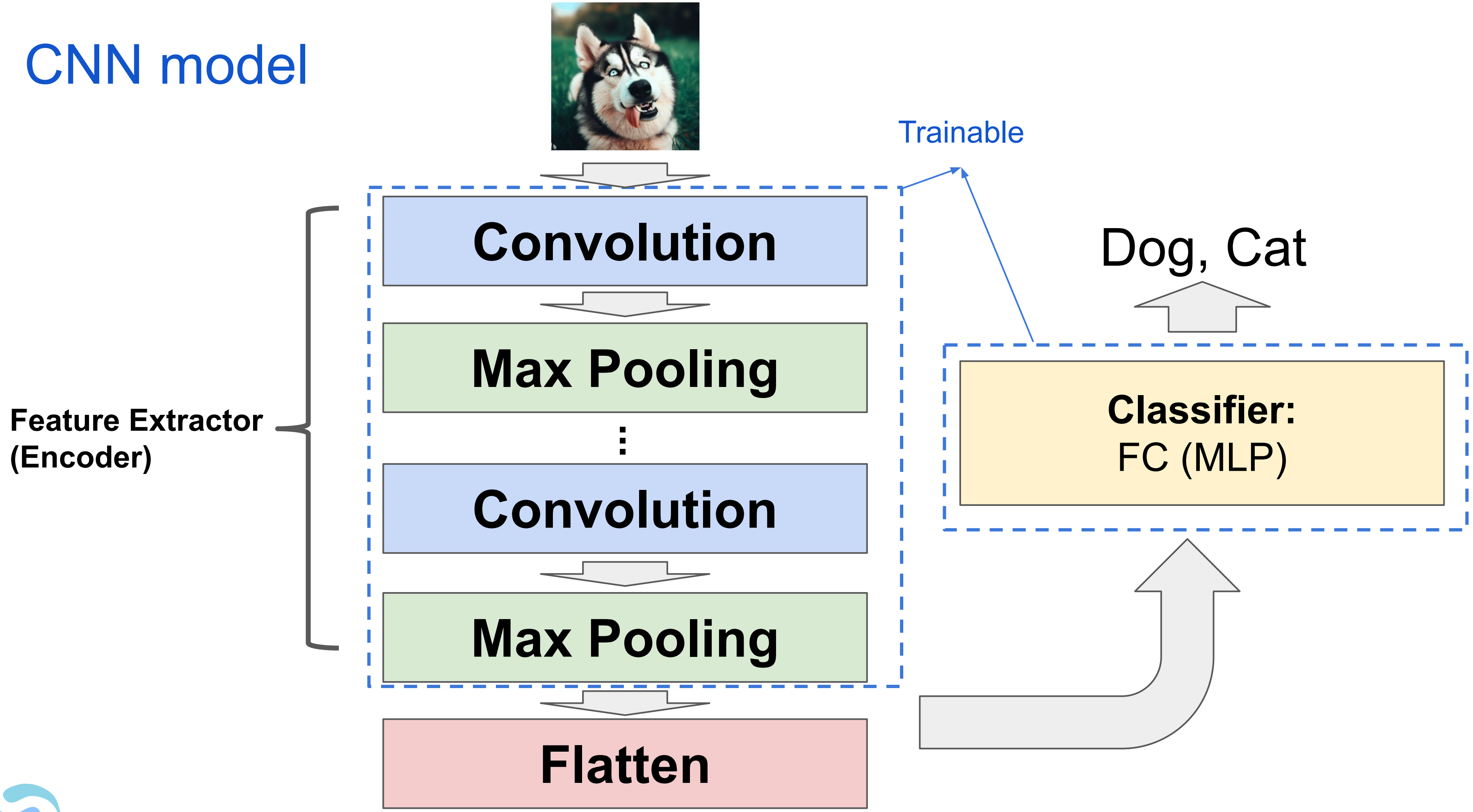
nn.Linear(8, 5)

nn.Linear(5, 5)

nn.Linear(5, 2)

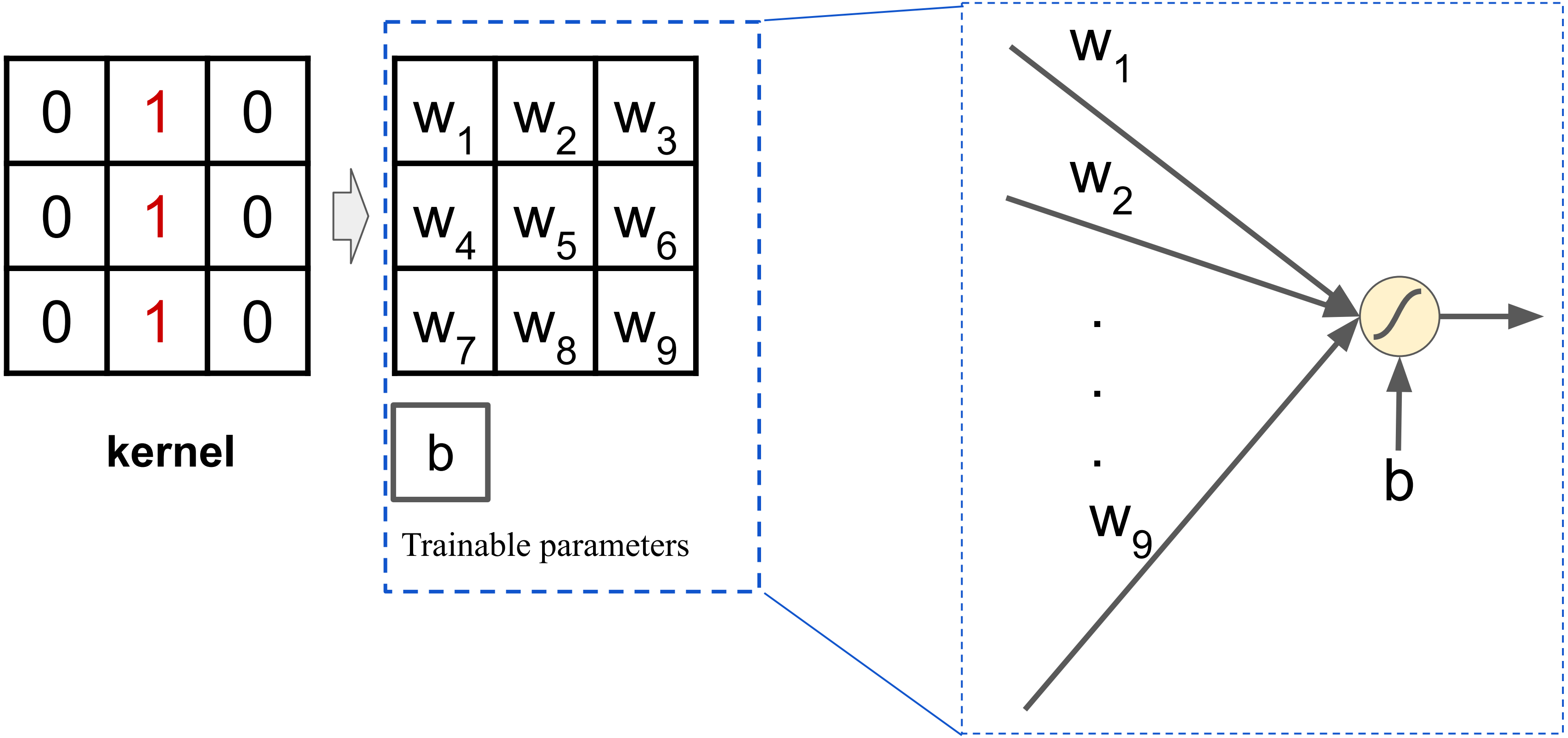
FUSION

CNN model



Trainable
Convolution
Max Pooling
⋮
Convolution
Max Pooling
Flatten
Feature Extractor
(Encoder)
Dog, Cat
Classifier:
FC (MLP)

Convolution Layer Parameters



Convolution v.s. Fully Connected (# of parameters)

(3, 100, 100)

Conv: 1, 3x3 filters

input channels

filter(kernel) count

$(3 \times 3 \times 3 + 1) \times 1 = 28$

filter
w x h x c

bias

FC: 1 neurons

neuron count

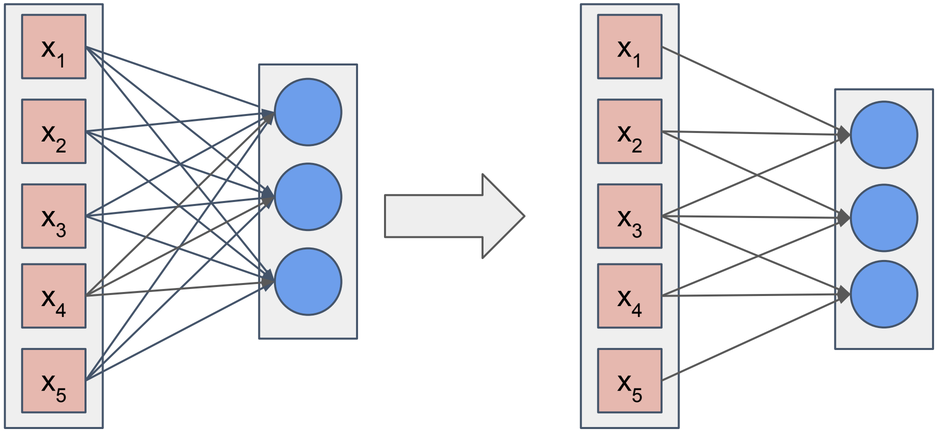
$(100 \times 100 \times 3 + 1) \times 1 = 30001$

# of pixels

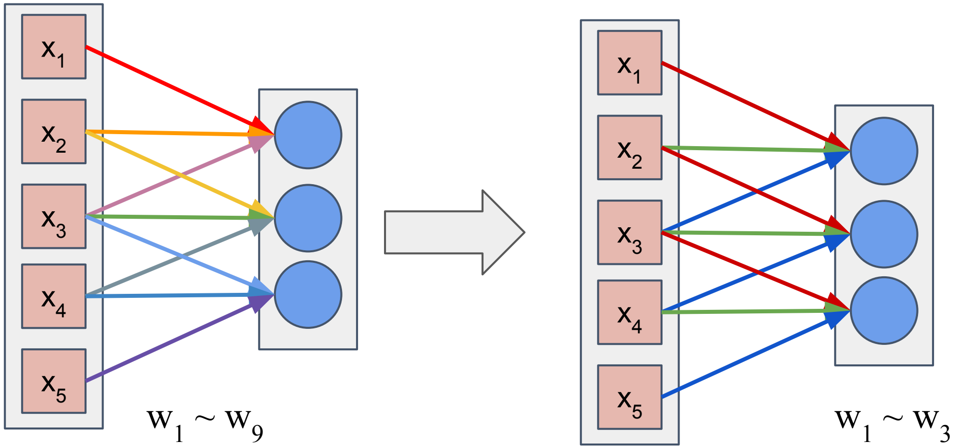
bias

Convolution

- Local connectivity
  - Neuron connect to **local features**



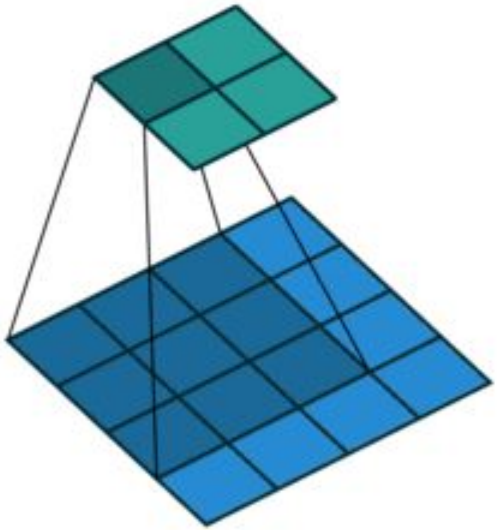
- Sharing parameters
  - filter use **same parameters** to convolve different region

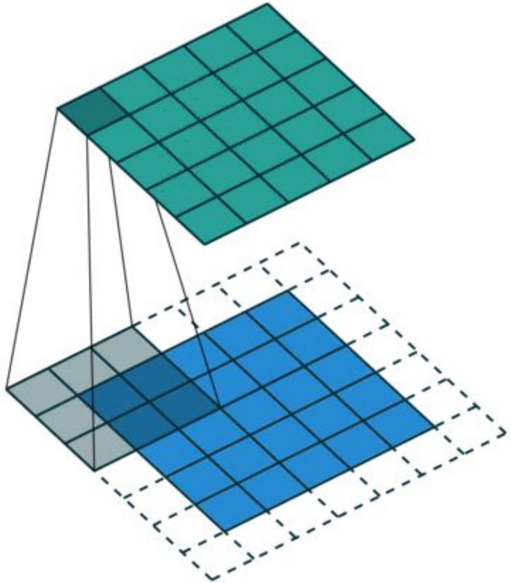


Convolution: Padding(填充)

- torch.nn.Conv2d(padding=0)
- Default: 0, no padding

- torch.nn.Conv2d(padding='same')
- torch.nn.Conv2d(padding=1)
- padding ≅ (kernel_size - 1) / 2
- same: input size = output size





source

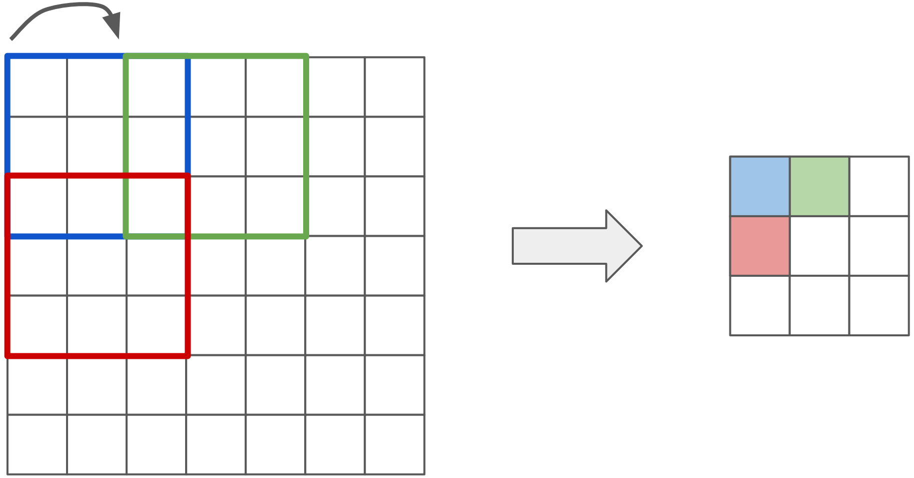
Stride (步長)

- Trainable **Pooling** layer
- Default: 1

PyTorch

```
torch.nn.Conv2d(C_1, C_2, strides=2)
```

strides = 2

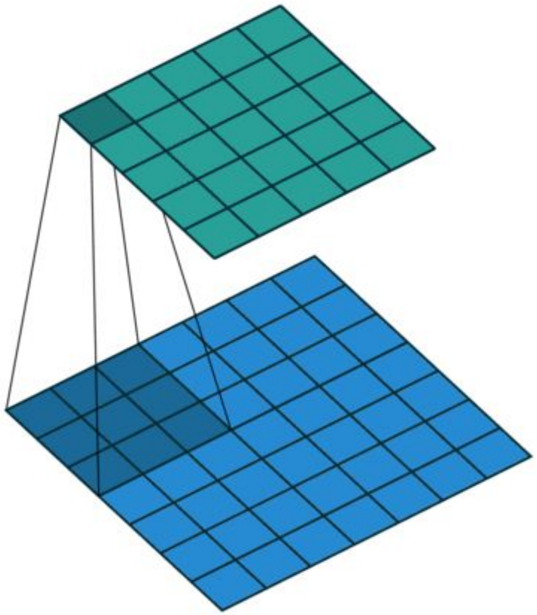


$(C_1, 7, 7)$

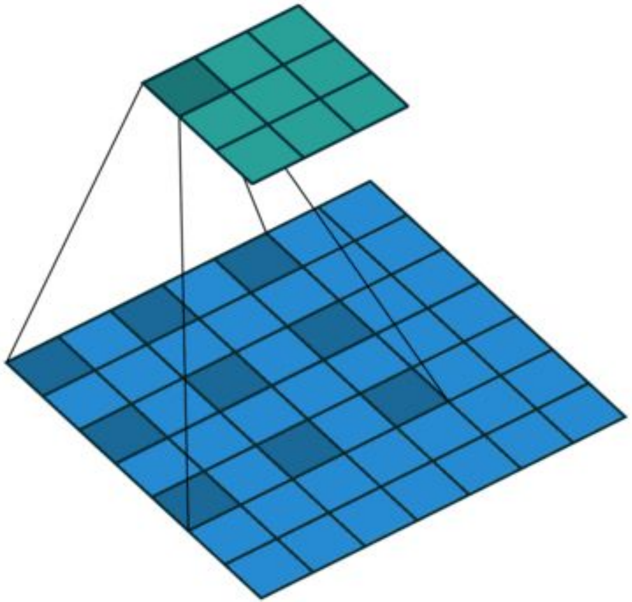
$(C_2, 3, 3)$
$\approx (C_2, H/2, W/2)$

Convolution: Dilation

- Increase the **receptive field**

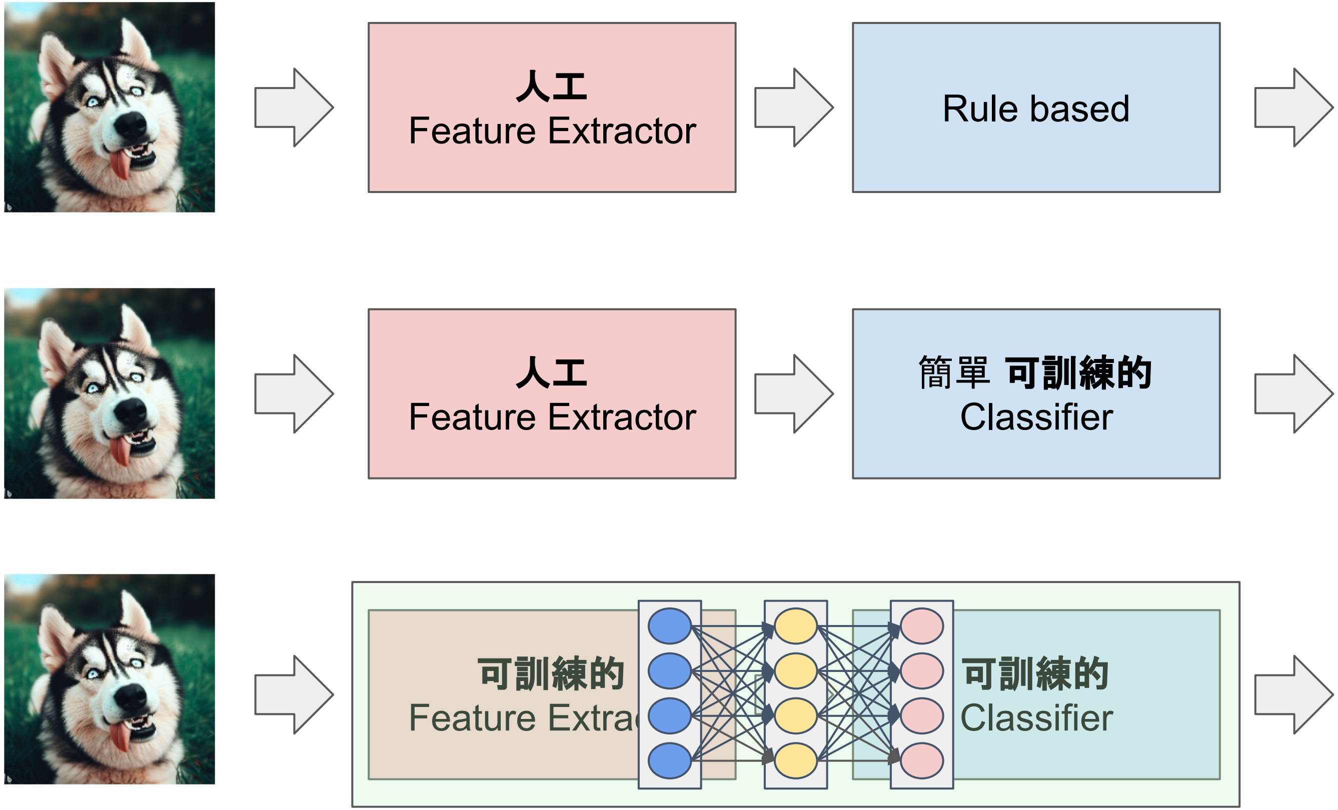


torch.nn.Conv2d(dilation=1)

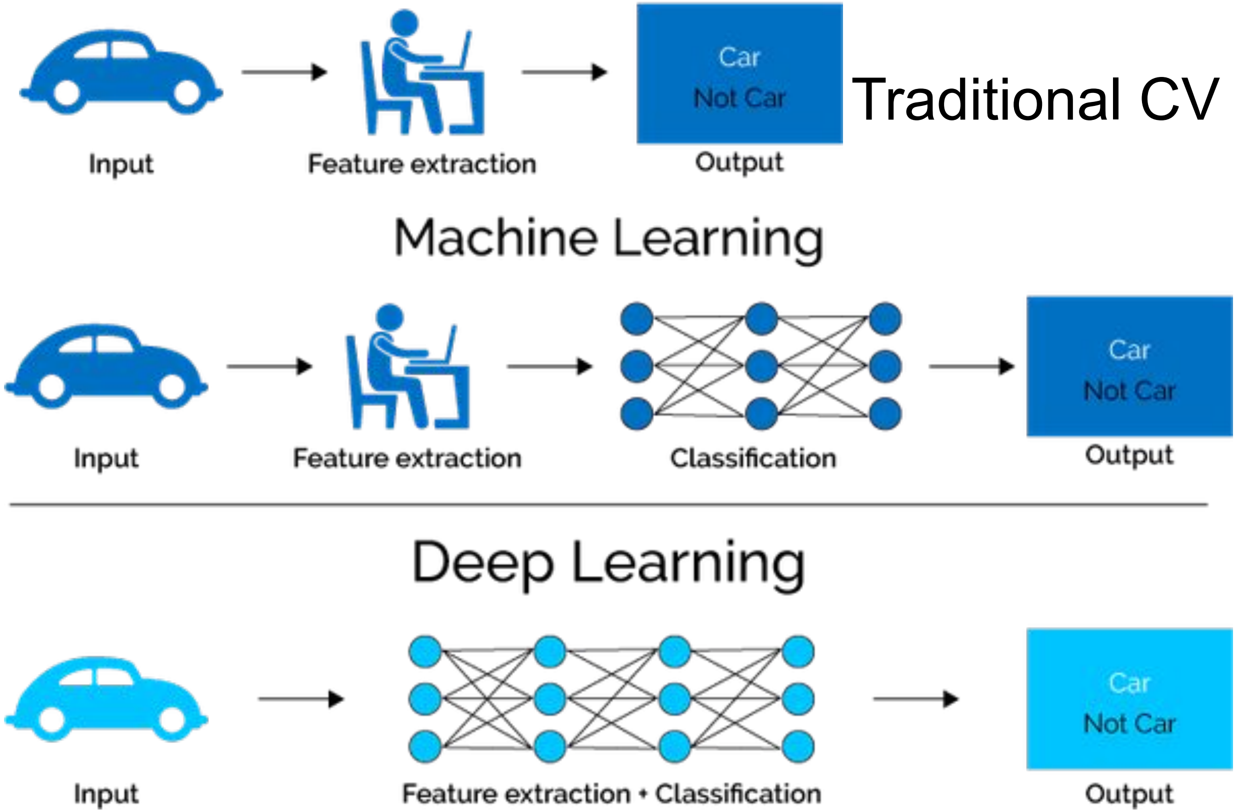


torch.nn.Conv2d(dilation=**2**)

Traditional v.s ML v.s DL



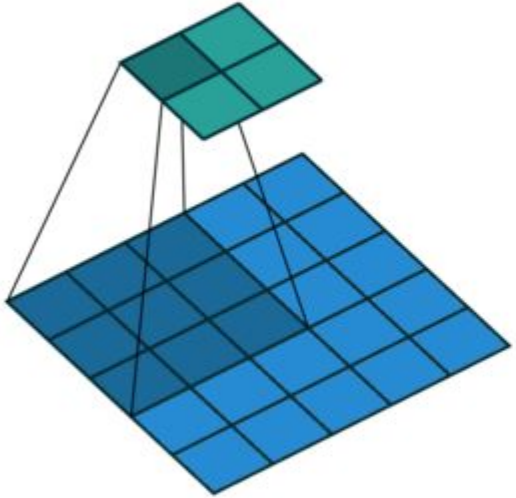
End-to-end Training

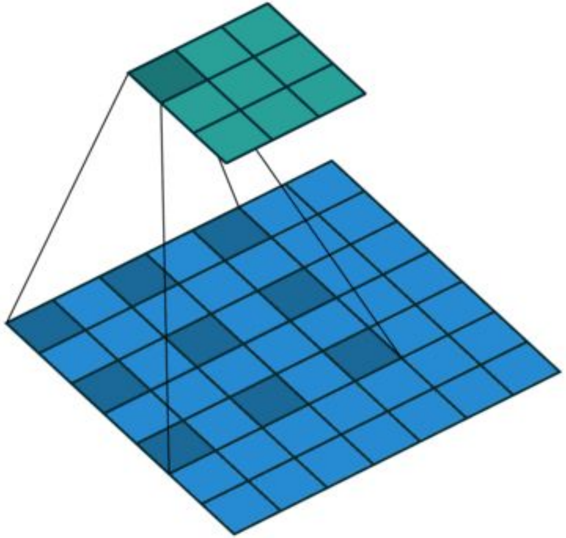


Traditional CV

Materials

- Convolution Animation
- https://github.com/vdumoulin/conv_arithmetic





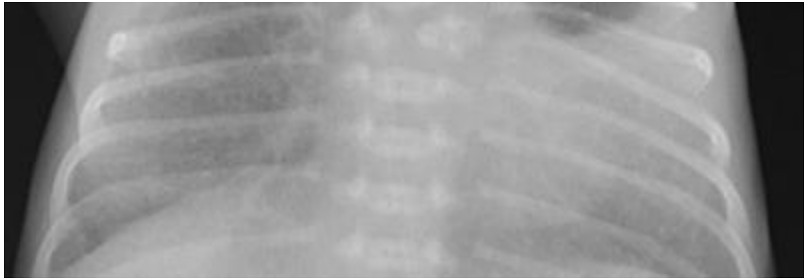
Summary

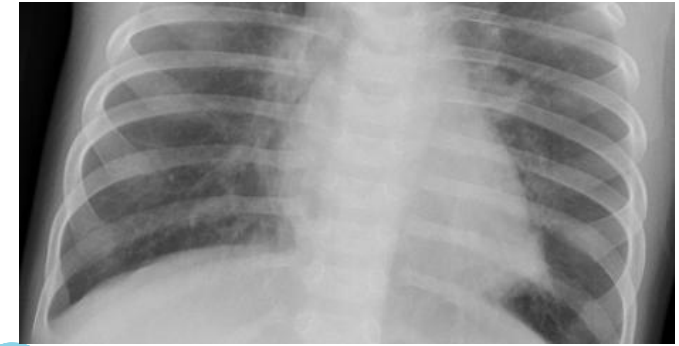
- 什麼是卷積層？
- 卷積神經網路架構
    - Conv.
    - Maxpooling
    - Flatten
    - Classifier
- 卷積神經網路參數

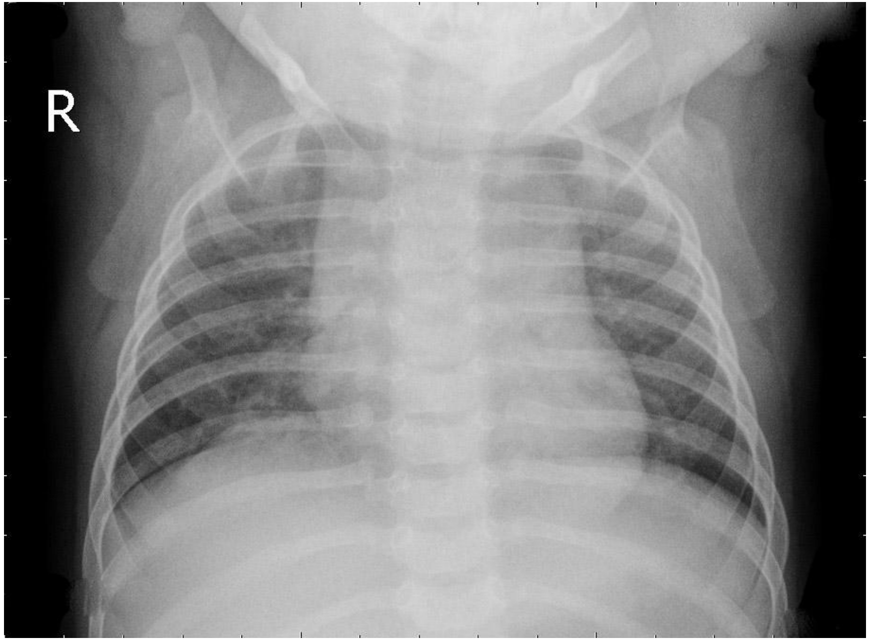
Yann LeCun

Convolutional Network Demo from 1993
https://youtu.be/FwFduRA_L6Q

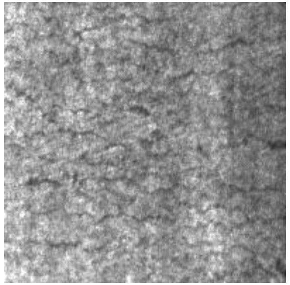
Exercise: Pneumonia Classification

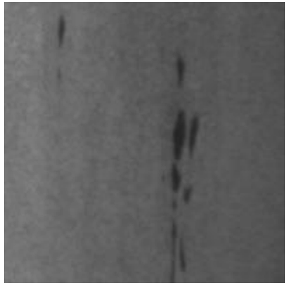


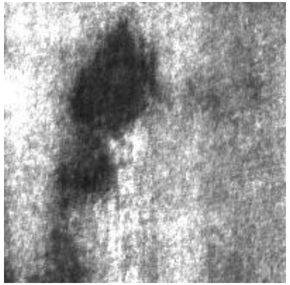


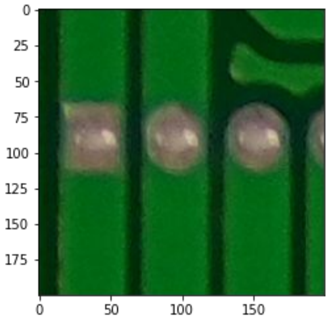


Exercise: Defect Classification



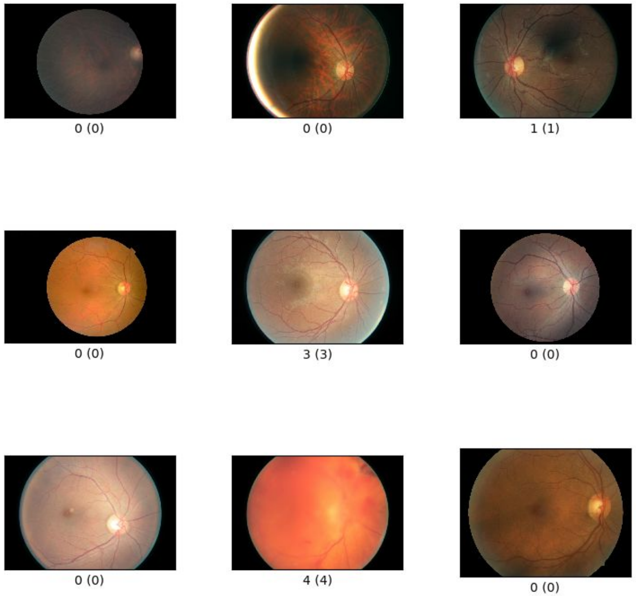






HW Retinopathy Classification

- Kaggle link:
- https://www.kaggle.com/c/diabetic-retinopathy-classification-3
- 5 classes classification: 0~4



Deep Learning Training Tips

Training Process

部署

Overfitting
過擬合

定義函式集(模型)

評分

N

N

Y

Y

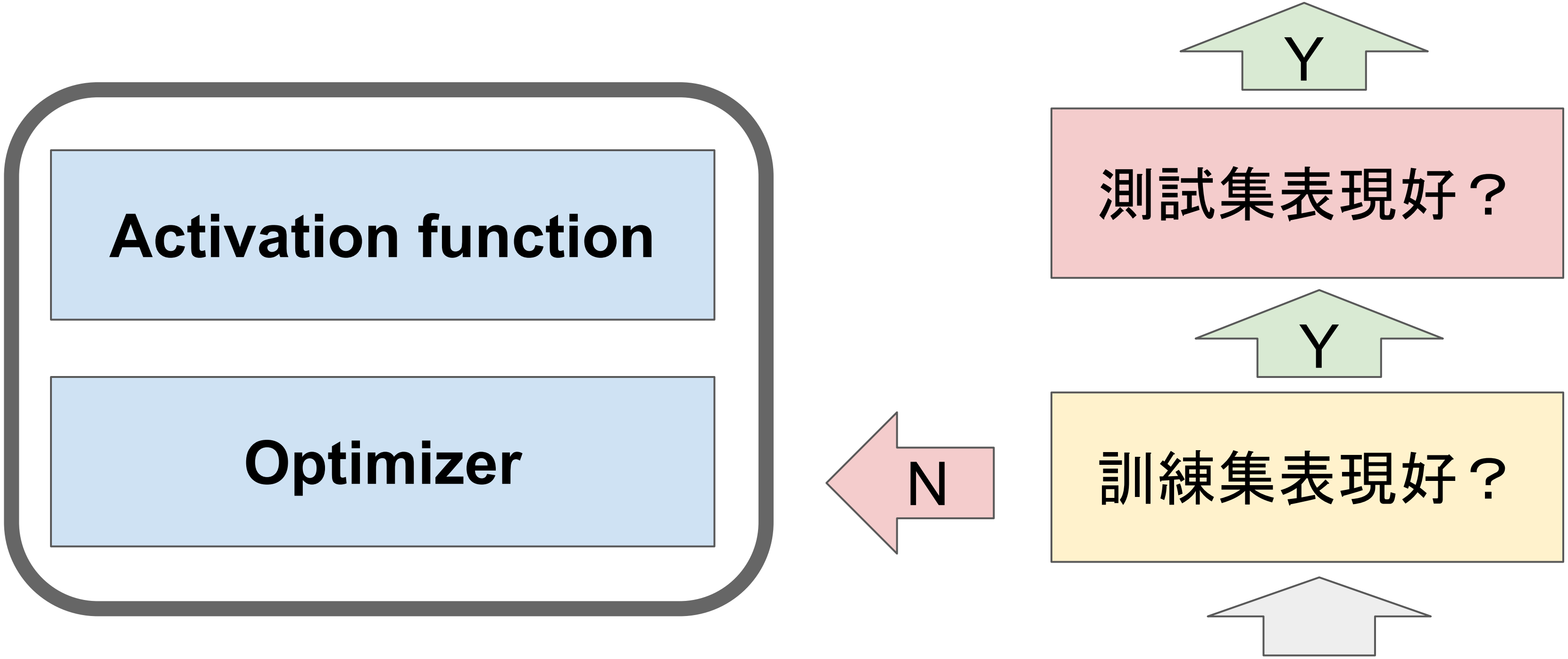
測試集表現好？

訓練集表現好？

ref: [李宏毅DL]

FUSION

Training Process

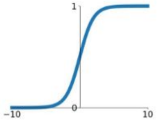


ref: 李宏毅DL

Activation Function

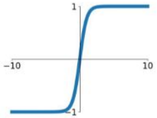
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



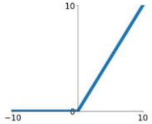
tanh

$\tanh(x)$



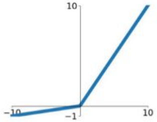
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

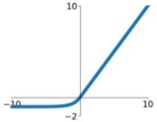


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



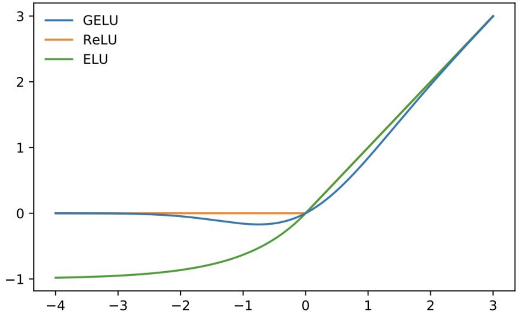
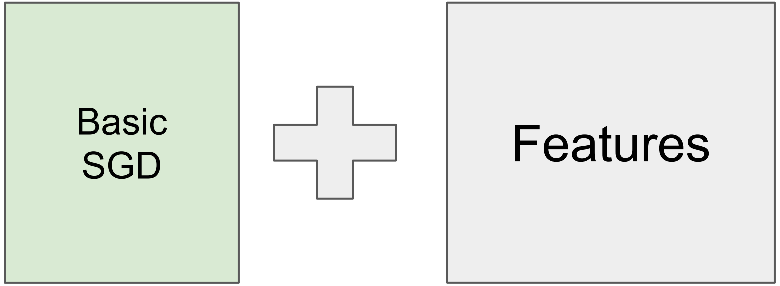


Figure 1: The GELU ($\mu = 0, \sigma = 1$), ReLU, and ELU ($\alpha = 1$).

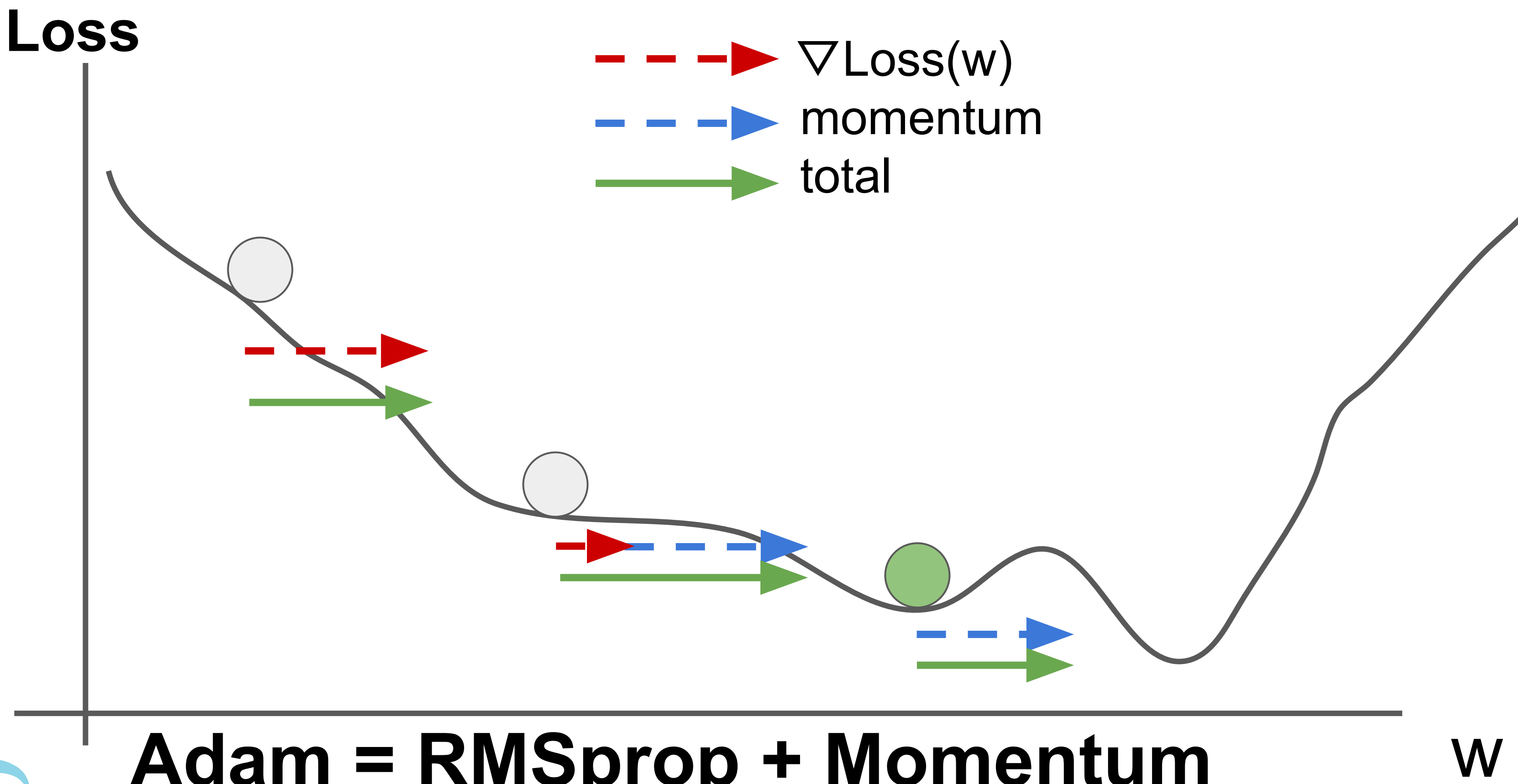
PyTorch
Sigmoid (0~1)
tanh (-1 ~ 1)
ReLU
LeakyReLU
ELU
GELU
Mish
…

Optimizer - Learning rate

- torch.optim
- SGD (Stochastic Gradient Descent)
- **Adam, AdamW**
- Adagrad (Adaptive Learning Rate)
- RMSprop
- **Ranger, Ranger21**

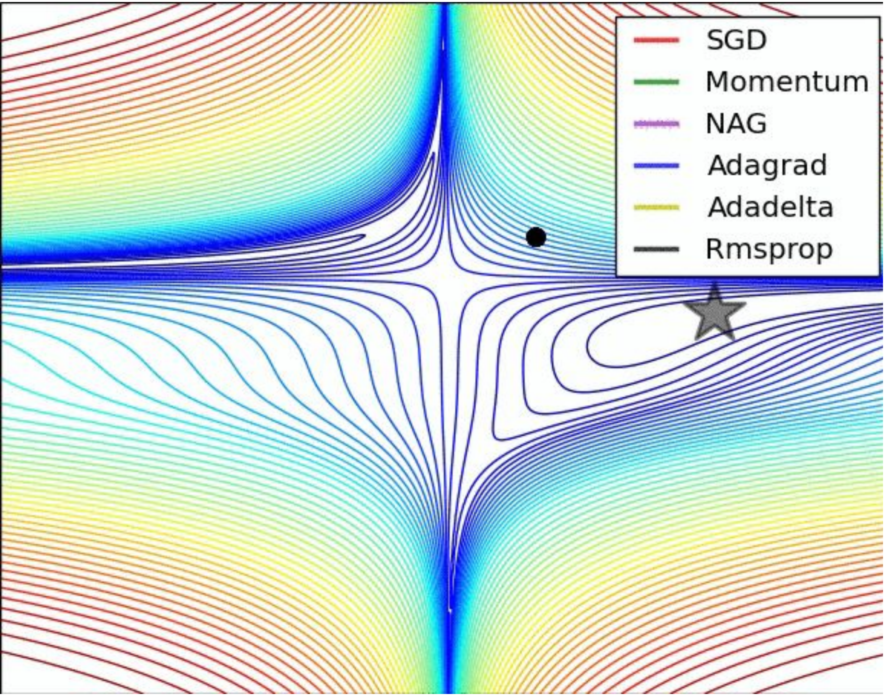


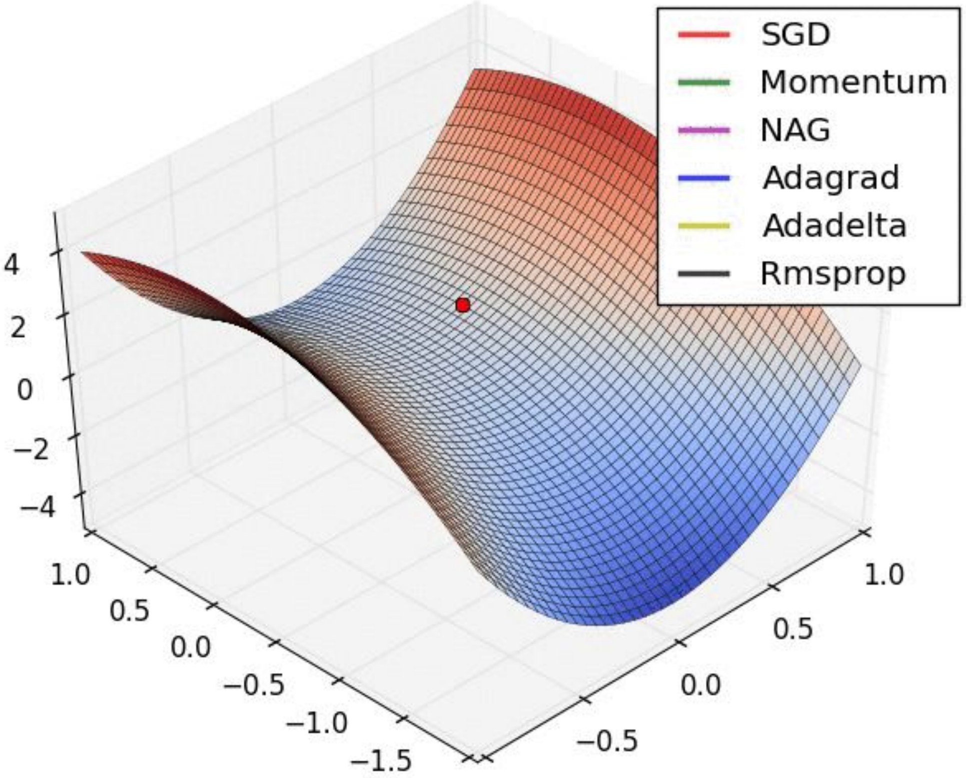
Momentum



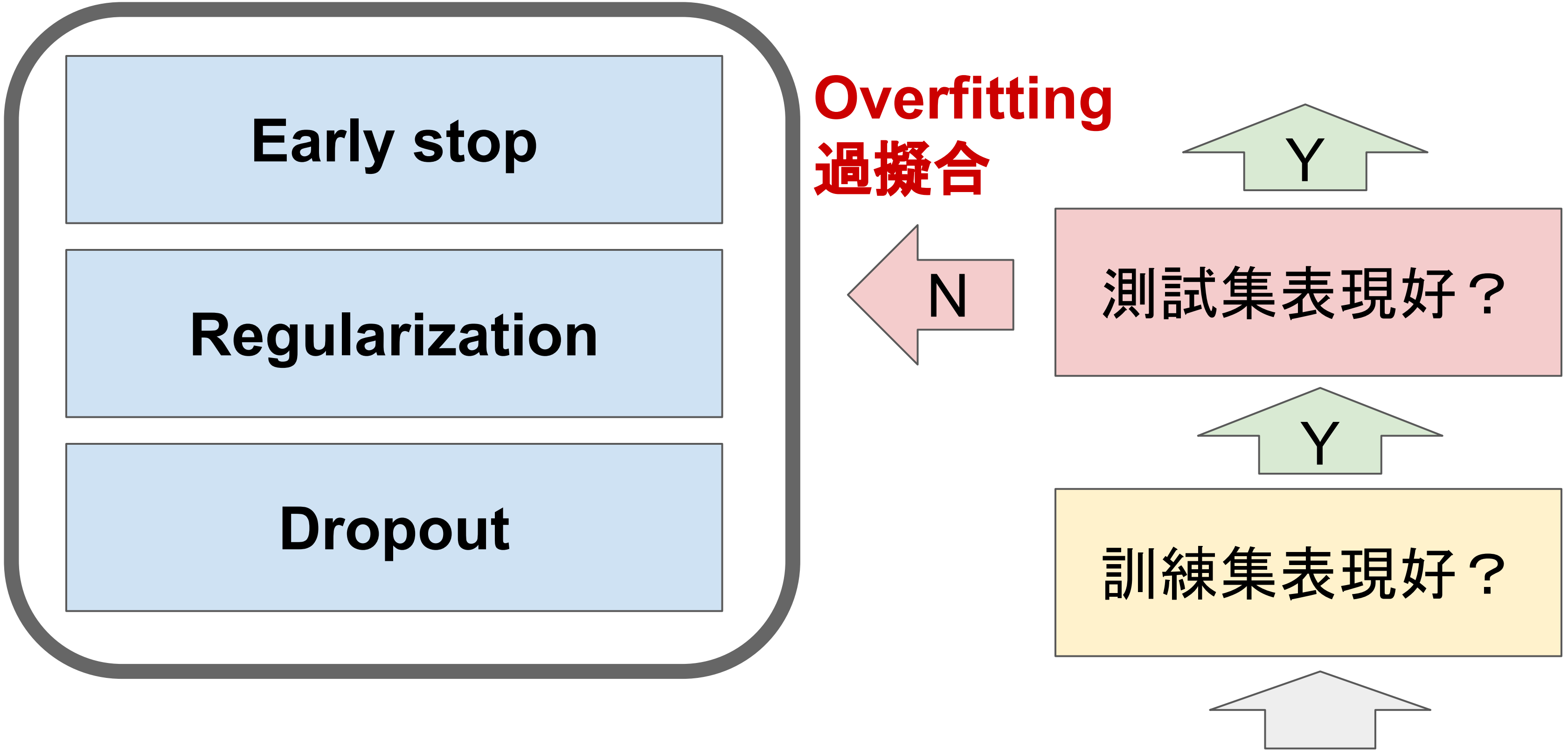
Adam = RMSprop + Momentum

Optimizer


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
4
2
0
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.5
1.0
0.5
0.0
−0.5

Training Process

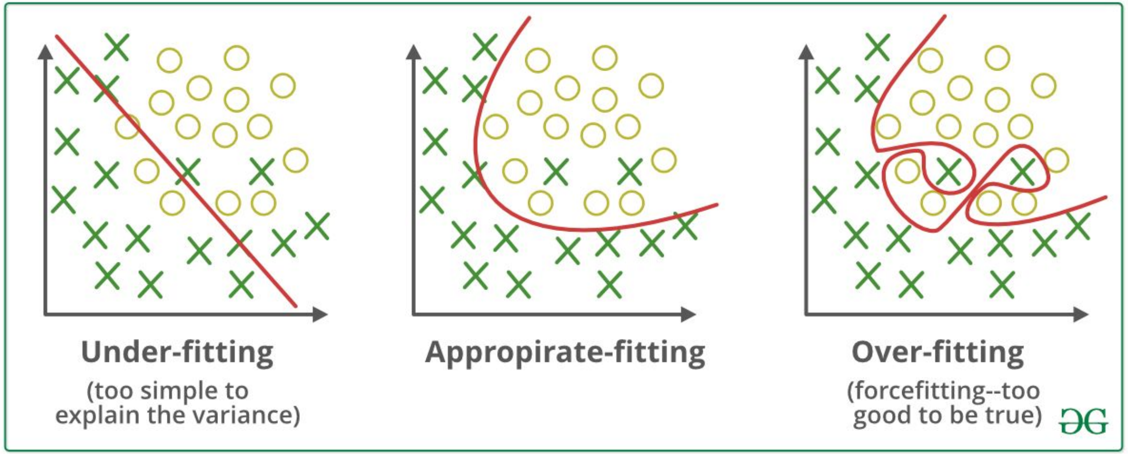


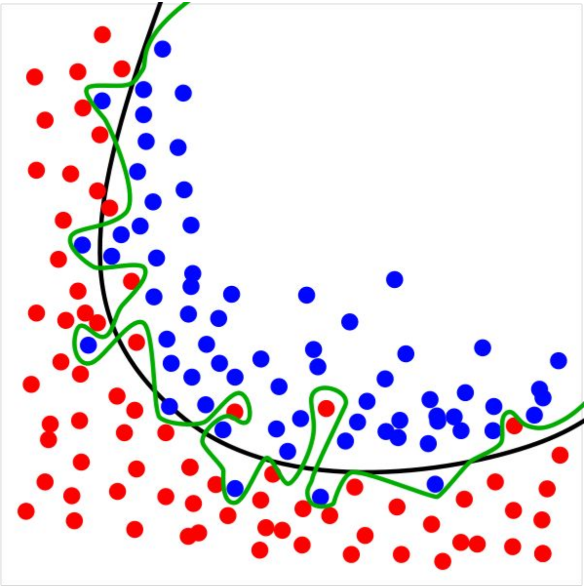
Overfitting (過擬合): Regression

Training loss ↓, Test/Validation loss ↑

The curse of deep learning!

Overfitting : Classification





Early stop

IF improved:

keep training

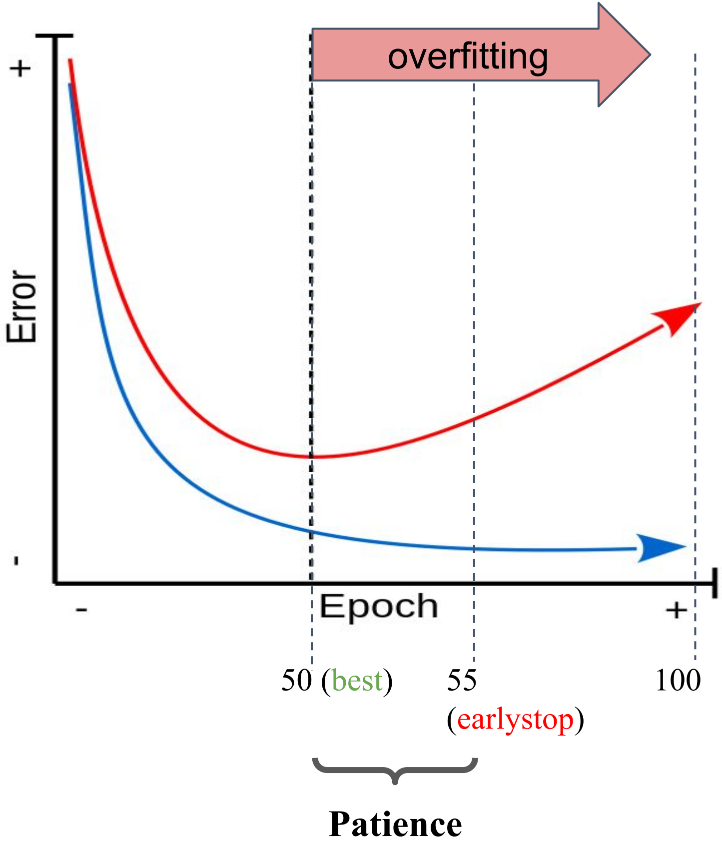
ELSE:

IF no improvement after **patience** epochs

stop training

ELSE

keep training



FUSION

Early stop

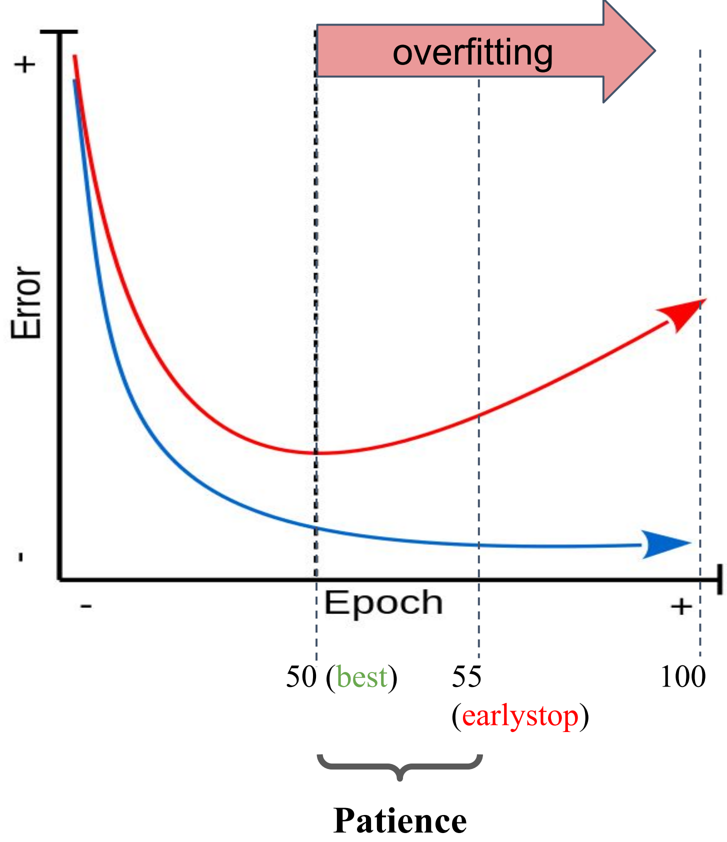
PyTorch

Error

```python
EPOCHS = 50
# Earlystopping
patience = 5
earlystop_counter = 0
best_loss = np.inf

for epoch in tqdm(range(EPOCHS)):
    train_loss, train_acc = train(dataloa
    val_loss, val_acc = test(dataloader_v
    # Earlystopping
    if val_loss < best_loss:
        earlystop_counter = 0
        best_loss = val_loss
    else:
        earlystop_counter += 1
    if earlystop_counter >= patience:
        print('Early stop!')
        break
```

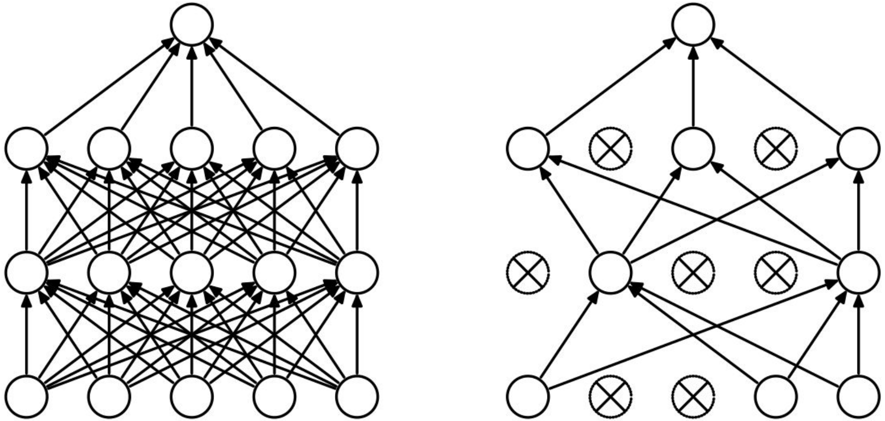
also save best model



FUSION

Dropout

- Sub-network with less parameters
- Reduce over fitting
- torch.nn.Dropout
- p: probability of an element to be zeroed. Default: 0.5



(a) Standard Neural Net

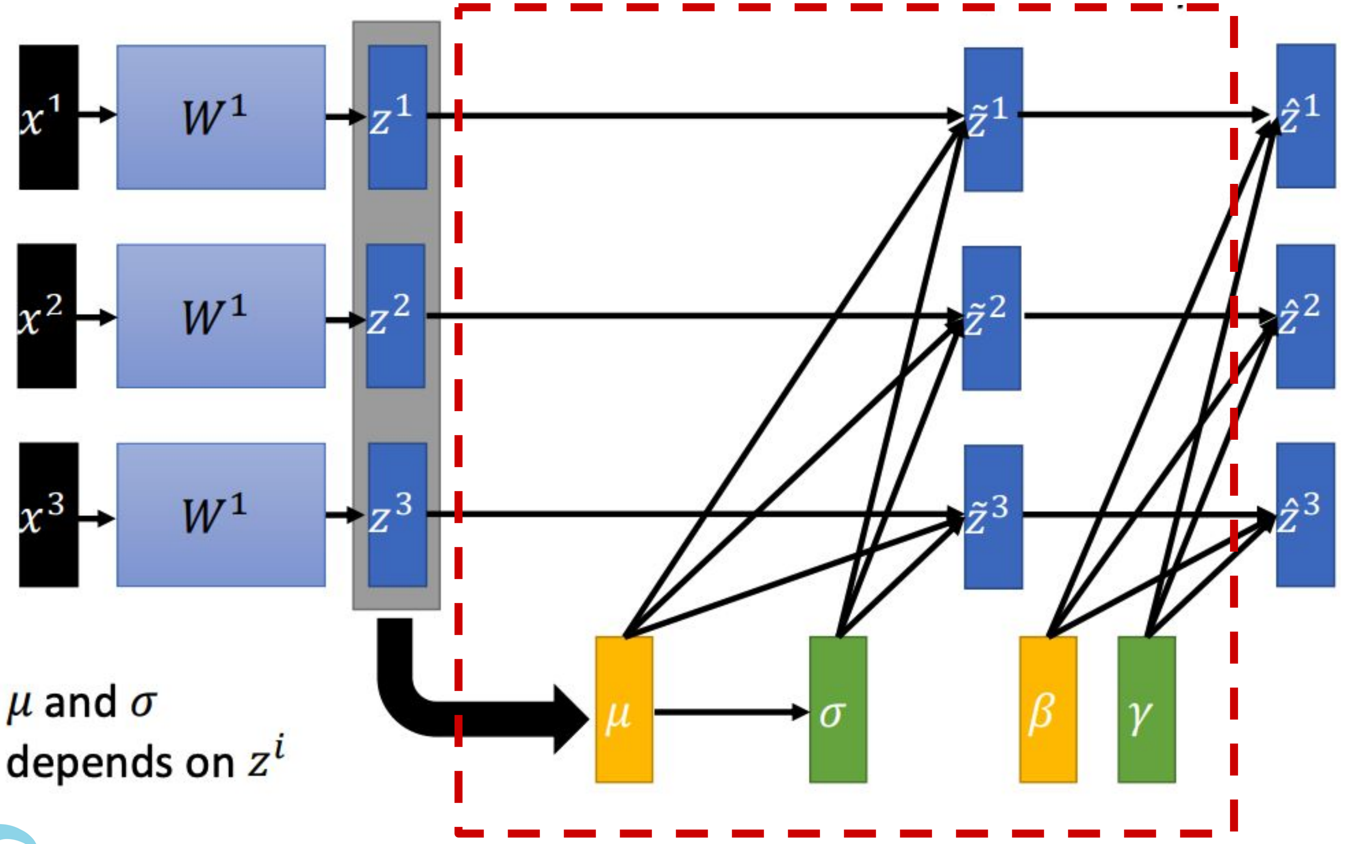
(b) After applying dropout.

Inference Training

PyTorch

```
classifier = nn.Sequential(
    nn.Linear(100, 64),
    nn.ReLU(),
    nn.Dropout(p=0.3),
    nn.Linear(64, 3)
)
```

Batch Normalization



$$\tilde{z}^i = \frac{z^i - \mu}{\sigma}$$

$$\hat{z}^i = \gamma \odot \tilde{z}^i + \beta$$

μ: mean
σ: std

γ, β: learnable params

source: 李宏毅DL

Normalizations Layers

- Batch Nomralization
- Layer Normalization
- Instance Norm
- Group Norm

PyTorch

```
nn.Conv2d(1, 16, 3),
nn.BatchNorm2d(num_features=16),
nn.ReLU(),
nn.Conv2d(16, 16, 3),
nn.BatchNorm2d(16),
nn.ReLU(),
```

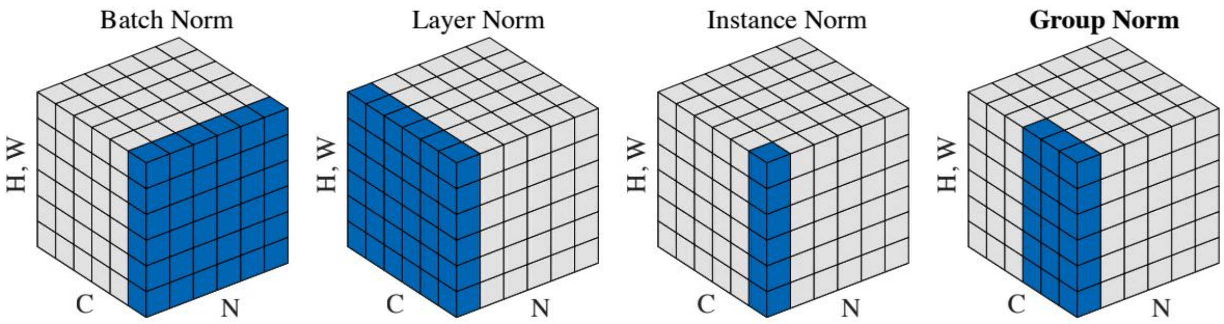
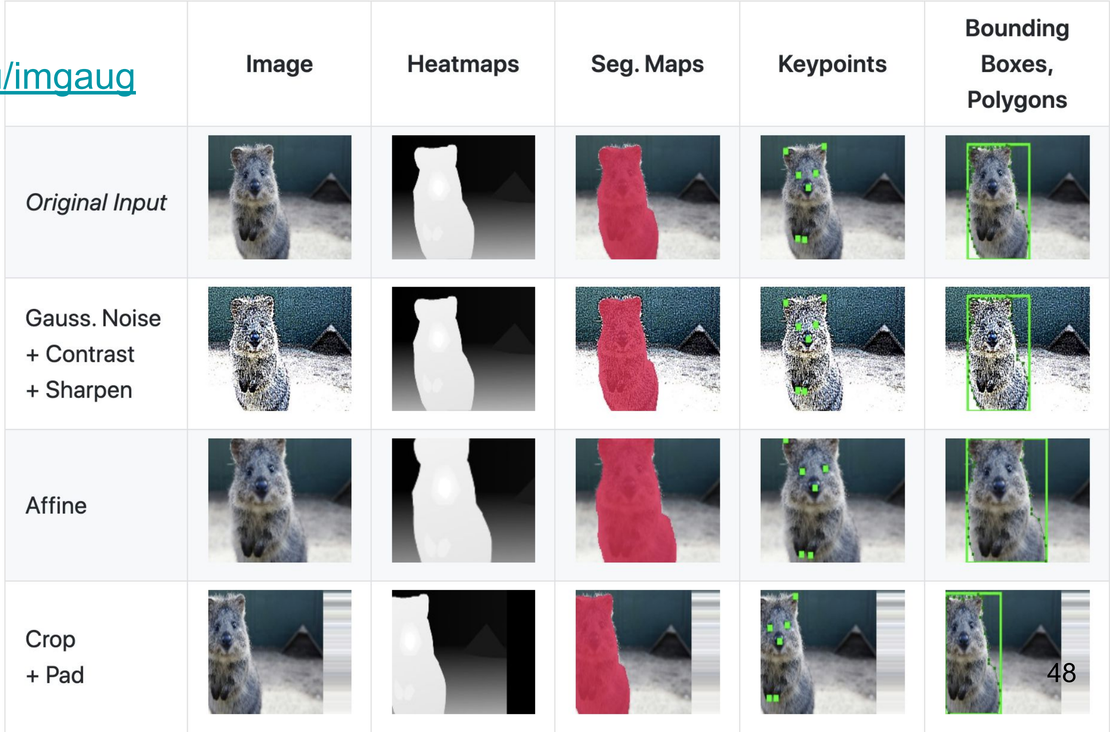


Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

Data Augmentation

1. Generate more data by some techniques
2. Make model more robust
3. Frameworks
    a. [torchvision.transforms]()
    b. imgaug: https://github.com/aleju/imgaug
        i. Sementation maps
        ii. bounding boxs
        iii. …
    c. [albumentations]()
    d. Others



imgaug

```python
# augmentation
seq = iaa.Sequential([
    iaa.Fliplr(0.5), # 50% horizontal flip
    iaa.Flipud(0.5), # 50% vertical flip
    iaa.Affine(
        rotate=(-10, 10), # random rotate -45 ~ +45 degree
        shear=(-16,16), # random shear -16 ~ +16 degree
        scale={"x": (0.8, 1.2), "y": (0.8, 1.2)} # scale x, y: 80%~120%
    ),
])
```

```python
# Augment 1 image
img_aug = seq.augment_image(img)
```

```python
# Augment images (batch size = 4)
img_batch = np.stack([img]*4) # (4, 60, 184, 3)
img_aug_batch = seq.augment_images(img_batch)
```

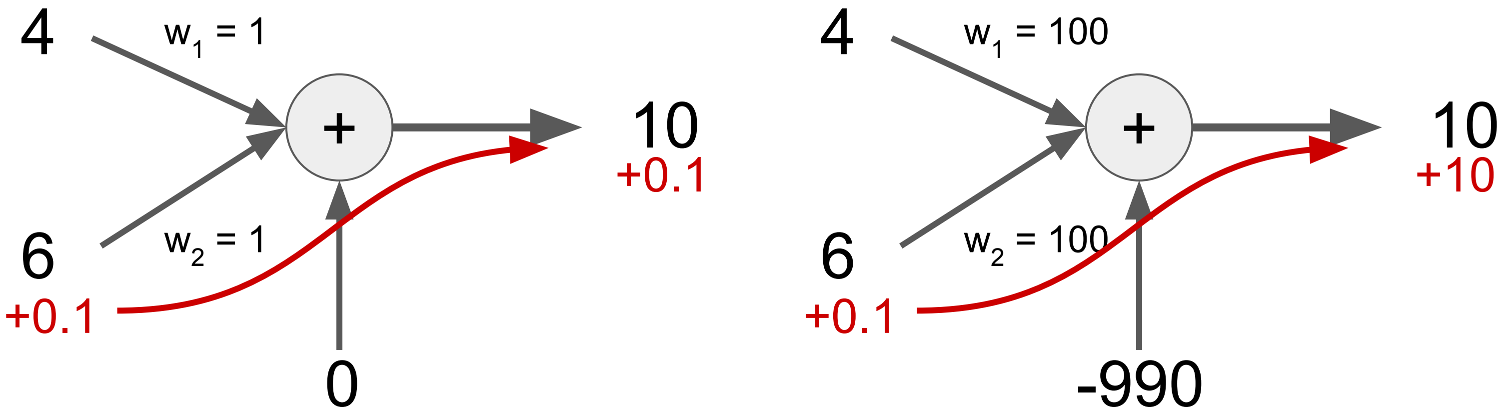
FUSION

torchvision.transforms

- Illustration of transforms
- Tensor transforms and JIT

Regularization (正則化)

- Reduce overfitting
- Reduce influence of **noise**



PyTorch [How to use L1, L2 and Elastic Net regularization with PyTorch?]